VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Určení komunit pomocí detekce hran spojujicích komunity

Community Detection by Betweenness-Based Methods

2014

Ondřej Svačina

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Ondřej Svačina**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Určení komunit pomocí detekce hran spojujicích komunity
Community Detection by Betweenness-Based Methods

Zásady pro vypracování:

Jednou z možností detekce komunit v komplexních sítích je použití algoritmů pro dělení grafů. Existuje však řada jiných přístupů, mezi nimi i přístupy využívající detekci hran spojujicích komunity. Cílem práce je výběr a implementace vhodných algoritmů, provední experimentů nad reálnými daty a interpretace výsledků.

1. Seznamte se s problematikou detekce komunit v komplexních sítích.
2. Prostudujte existující přístupy k detekci komunit pomocí detekce hran.
3. Vyberte vhodné metody a vhodná testovací data.
4. Vybrané metody naimplementujte, proveďte experimenty a jejich výsledky vyhodnoťte.

Seznam doporučené odborné literatury:

[1] M. E. J. Newman, Networks: An Introduction, Oxford University Press (2010), ISBN-10: 0199206651.
[2] Dle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **RNDr. Eliška Ochodková, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ve Studénce Dne: 26.4.2014 Podpis: 

Rád bych poděkoval paní RNDr. Elišce Ochodkové, Ph.D. za trpělivost, odbornou pomoc a konzultaci při vytváření této práce.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Poděkování

Tato práce byla vypracována s podporou projektu Rozvoj lidských zdrojů ve výzkumu a vývoji moderních soft computingových metod a jejich praktického využití, reg. č. CZ.1.07/2.3.00/20.0072 podpořeného Operačním programem Vzdělávání pro konkurenceschopnost, financovaného ze strukturálních fondů EU a státního rozpočtu ČR.

Abstrakt

Tato diplomová práce se zabývá rozpoznáváním komunit pomocí detekce hran spojujícími komunity. Cílem práce bylo si nastudovat současně používané algoritmy, vybrat z nich ty nejpoužívanější a poté je analyzovat a vyhodnotit. Byly vybrány následující algoritmy: metoda nejkratší cesty, metoda náhodné procházky, metoda vzdálenost k zóně a Raddichiho algoritmus. Bylo naimplementováno několik verzí těchto algoritmů a poté budou výsledky vybraných verzí podrobeny analýze a vyhodnocení.

Klíčová slova: zóna, graf, komunita, shluk, metoda nejkratší cesty, metoda náhodné procházky, Raddichi, betweenness

Abstract

This thesis deals with community detection by betweenness-based methods. The aim of thesis was to study recent algorithms, selecet most used ones and then analyse them. Selected algorithms were: shortest-path method, random-walk method, distance to zone method and Raddichi algorithm. These algorithms were implemented in various versions and then results of selected versions were tested, analysed and evaluated.

Keywords: zone, community, shortest-path, random-walk, DTZ, distance to zone, Radicchi, betweenness

Seznam použitých zkratek a symbolů

DTZ	–	Distance to zone
GPS	–	Globální polohovací systém
Pan	–	Personal area network
Lan	–	Local area network
Wan	–	Wide area network

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V současné době vzrůstající obliba různých sociálních sítí má za následek, že analýza sítí je stále důležitější a užitečnější vědeckou disciplínou. Sociální sítě si lze představit jako graf, kde jsou jednotlivé osoby reprezentovány jako uzly a spojnice mezi těmito uzly jsou vztahy mezi osobami. Tento vztah může být například rodinné členství, přátelství nebo společné zájmy či aktivity. Sociální sítě však nejsou jediné zkoumané sítě. Existuje velké množství druhů sítí, například biologické sítě či dopravní sítě. Více o druzích sítí v kapitole 2.

Analýza sítí slouží k získání užitečných informací ze sítí. Pouhým pohledem na grafickou reprezentaci sítě se můžeme dozvědět pouze velice limitované množství informací, v sítích obsahujících desetitisíce uzlů a hran se dozvíme o celku pohledem ještě méně. Sítě se proto zpracovávají strojově a bylo zavedeno velké množství algoritmů zkoumající četné vlastnosti sítí. Zkoumané vlastnosti sítí se liší podle toho, jaký je náš cíl při zkoumání sítě. Například u sociálních sítí, provozovatele reklam a vyhledávače falešných účtů nás budou zajímat jiné vlastnosti sítě.

Jeden z důležitých nástrojů pro analýzu sítí a zároveň objekt zkoumání této práce jsou algoritmy vyhledávající komunity. Jedná se o algoritmy, které pomocí různých přístupů přerozdělí uzly sítě do jednotlivých skupin, které mají mnoho společného. Taková skupina může reprezentovat rodinu, klub či zboží s podobnými znaky. Existuje spousta přístupů k vyhledávání komunit. Více o tomto tématu v kapitole 3.

Cílem této práce bylo nastudovat současný stav jednoho z přístupů k vyhledávání komunit, a to pomocí detekce hran oddělující komunity. Dalším cílem je z těchto algoritmů vybrat nejpoužívanější, ty naimplementovat, a poté porovnat jejich rychlost a kvalitu výsledných rozdělení do komunit.

V této práci se nejdříve seznámíme s tím, jaké existují druhy sítí, jak jsou tyto sítě reprezentovány a co jsou to komunity. Dále bude probráno, jaké algoritmy pro nalezení komunit existují. V další kapitole budou podrobněji popsány algoritmy, jenž fungují na principu nalezení hran oddělujících komunity. Poté bude předvedeno, jakým způsobem byly tyto algoritmy naimplementovány. Nakonec budou provedeny testy porovnávající výsledné rozdělení uzlů do komunit, pomocí porovnání s vybraným referenčním rozdělením.

2 Sítě

Začneme tím, že si vysvětlíme, co to síť vlastně je. Síť je množina entit, jenž mohou být propojeny nějakou vazbou. Co tyto entity a vazby představují, záleží na typu sítě.

Jak bylo zmíněno v úvodu, sítě se podle oblasti použití dělí na množství kategorií. Do použití sítí a teorie grafů obecně patří jak jednoduché úlohy pro žáky základních škol, tak analýza zákazníků, kde na správné analýze závisí úspěch či krach velkých firem. Sítě se využívají v různých odvětví, například v marketingu, biologii, kde zkoumáme zkoumání DNA a enzymy, dopravě, informatice či energetice.

Asi nejjednodušší síť pro představu je dopravní síť, kde analyzujeme vytíženost jednotlivých cest a počty semaforů a křižovatek. Díky těchto údajů můžeme naplánovat ideální cesty na převoz zboží, které teoreticky potrvají nejkratší dobu. Stejně tak může správa měst analyzovat automobilovou dopravu. Pokud se v daném úseku často vyskytují dopravní zácpy, můžeme vytvořit novou cestu vedoucí k cíli přes místa s nejnižším provozem pro maximální efektivitu.

V této kapitole budou rozepsány nejvýznamnější typy sítí, dále bude vysvětleno co je to graf a jakým způsobem reprezentuje sítě a jaké jsou nejvýznamnější vlastnosti grafů. Nakonec bude podrobně vysvětleno, co to jsou komunity, o jejichž vyhledávání bude zbytek této práce.

2.1 Druhy sítí

2.1.1 Dopravní sítě

Jak již bylo naznačeno, teorie sítí má využití v dopravě. Příkladem takové sítě může být síť, kde vrcholy jsou města a hrany jsou cesty mezi městy. Pod pojmem doprava si nesmíme představit pouze dopravu po silnicích. Pod pojmem doprava se skrývá například i vlaková, lodní nebo letadlová doprava. Ve všech těchto možnostech lze sítě využívat. Více o dopravních sítích v textu [1].

Mezi problémy, které řeší dopravní sítě, můžeme zařadit následující problémy. První problém je zajištění plynulého chodu dopravy, aby nám nevznikla dopravní tepna, kterou bude potřebovat projet daleko vetší počet vozidel, než je tato cesta schopna pojmout. V takovém případě musí buď vzniknout alternativní cesta, nebo se musí tato cesta rozšířit. Avšak takový problém by nevznikl, kdyby byla dopravní síť správně analyzovaná a bylo dopředu odhadnuto, jak silný provoz v tomto místě vznikne.

Druhý problém se týká přepravy zboží. Dopravní společnost potřebuje zajistit co nejrychlejší a přitom nejlevnější přepravu zboží pro maximalizaci svých zisků. Musí se brát ohled nejen na celkovou délku trasy, ale i na další vlastnosti jako kvalita silnice, clo, zda tudy vůbec mají povoleno projet, jak často bývá silnice ucpaná a jiná rizika.

Se třetím problémem se setkávají často obyčejní řidiči, a to s navigací pomocí GPS. Pro řidiče je pohodlné, pokud jim systém automaticky vyhledá ideální cestu do místa určení, ale i to musí být nějak spočítáno. Přístroj pro navigaci má jen omezený výkon a paměť, tudíž musíme výpočty optimalizovat. Jeden ze způsobů je pro hledání cesty mezi městy nahradit jednotlivá města body, tudíž ignorovat sít cest uvnitř měst, a až poblíž města

spočítat, jak projet samotným městem. Dále mapy nemusí obsahovat informace o málo významných cestách nebo můžou obsahovat chyby, proto i při jízdě s navigací si musíme dávat pozor.

2.1.2 Sociální sítě

Se sociálními sítěmi se lidé setkávají prakticky denně. Příkladem sociální sítě může být síť taková, kde vrcholy představují osoby a hrany představují jejich přátelství. Pod pojmem sociální síť si nemusíme představit pouze sítě komunikační, jako je například poslední dobou velice rozšířený Facebook. Sociální síť může představovat téměř jakoukoli lidskou interakci. Mnoho lidí, aniž by věděli cokoli o teorii sítí, sami takovou síť vytvořili, a to rodokmen. Rodokmen je dostatečně malá a přehledná síť, abychom z ní dokázali přečíst námi hledané informace i bez použití analýzy, ale ne vždy je tomu tak.

Sociální sítě jsou sociální struktury vytvořené svými účastníky, již můžou být například osoby či organizace, kteřé mohou být vzájemně propojováni. Analýza sociálních sítí poskytuje množství nástrojů pro zkoumání vlastností sítí, či vzorů v sítích. Sociální sítě jsou často využívány pro mapování lidského chování, avšak zdaleka nejvíce jsou využívány komerčně, a to na cílenou reklamu.

Jen málokdo nezažil reklamy na internetu. Avšak reklamy na zboží, o které nemá osoba vůbec zájem, jsou pouze vyhozené peníze. A právě tyto informace společnostem zajišťuje analýza sociálních sítí. Informace o navštívených stránkách, dříve nakoupeného zboží, či informace uvedené na profilech na internetu, to vše může být analyzováno. Výsledkem této analýzy může být informace, o jaké zboží by mohl daný člověk mít zájem. Firma má větší šanci na zisk zákazníků a lidem se zobrazují reklamy na zboží, které by je vážně mohlo zajímat. Více o sociálních sítích v knize [2].

2.1.3 Počítačové sítě

Počítačové sítě jsou sítě umožňující přenos dat mezi jednotlivými počítači. V této síti jsou uzly obvykle tvořené jednotlivými počítači a hrany představují propojení těchto počítačů. Pro přenos dat může být buď použita linka, nebo bezdrátové připojení. Nejznámější počítačová síť je Internet.

Počítačové sítě mohou být podle rozlohy rozděleny do následujících kategorií. Pan (Personal area network) je síť do deseti metrů a obvykle se jedná o tiskárny, telefony či fax. Lan (Local area network) je síť počítačů propojených obvykle v rámci jedné budovy. Jedná se o přístroje, které spolu často komunikují. Nakonec uvedu WAN (Wide area network). Jedná se o síť velké rozlohy, může se jednat o celé kontinenty.

Analýza počítačových sítí může být použita pro zajištění optimální rychlosti přenosu dat, spolehlivosti přenosu dat, či zajištění, aby celá síť počítačů nebyla zneschopněna z důvodu výpadku jednoho počítače.

2.1.4 Biologické sítě

Biologické sítě jsou veškeré sítě, které se věnují živočišným druhům, či potravním řetězcům. Cílem těchto sítí je pochopení vztahů v ekologii, evoluci a fyziologických studiích, jako jsou neuronové sítě. Spousta komplexních biologických systémů může být vymodelováno do podoby sítí.

Následují příklady některých biologických sítí.

Sítě interakce mezi proteiny jsou nejčastěji zkoumané biologické sítě. Například bylo zjištěno, že silně propojený protein je významnější pro přežití než méně propojené. Toto napovídá, že celková sít proteinů, a ne jednotlivé interakce má vliv na fungování organismů.

Potravní řetězce. Veškeré organismy jsou mezi sebou propojeny skrz potravu. Potravní řetězce mají značný vliv na ekologii. Kdyby měl nějaký živočišný druh vymřít, můžeme odhadnout, jaké by to mělo následky na zbytek organismů. Tato problematika je teď významná díky vymírání živočišných druhů díky změnám klimatu.

Sítě mezidruhové interakce. Jedná se o studium vzájemné pomoci či soupeření mezi živočichy. Například můžeme zjistit, jaké rostliny si navzájem pomáhají a zvýšit tak výnosy potravin. Více o biolgických sítí se dá dočíst v článku [3].

2.1.5 Sémantické sítě

Sémantické sítě se snaží zobrazit logický postup, či tok myšlenek. Typickým případem sémantické sítě je síť slov a jejich významu, či síť odvození nových slov.

2.2 Grafy

Nyní když víme, jak důležité sítě jsou, si ukážeme, čím jsou sítě vyjádřeny. Nejtypičtějším způsobem vyjádření sítí jsou grafy. Většina následujících definicí byla čerpána ze stránek [4] a anglických stránek [5]. Nyní si vysvětlíme některé základní pojmy okolo grafů a pojmy potřebné pro tuto práci.

Graf je skupina vrcholů, kde jednotlivé dvojice vrcholů můžou či nemusí být propojeny hranou.

Definice 2.1 *Graf je dvojice množin G(V, E), kde V je neprázdná množina vrcholů a E je množina hran, tvořená dvojicemi vrcholů.*

Orientovaný graf vznikne z grafu neorientovaného tím, že jeho hranám přiřadíme libovolnou orientaci. Typickým příkladem pro orientovaný graf může být graf toku proudu vody v potrubí, kde je důležité, kterým směrem voda teče.

Definice 2.2 *Graf G(V,E) je orientovaný, pokud je množina hran složena z uspořádaných dvojic uzlů. Graf G je neorientovaný, pokud je množina hran složena z neuspořádaných dvojic uzlů.*

Další běžný typ grafu je ohodnocení grafu. Grafy se dělí na ohodnocené a neohodnocené. U hranově ohodnocených grafů mají hrany určenou hodnotu tohoto spojení. Může se jednat o sílu vztahu, či délku propojení.

Definice 2.3 *Pokud je každé hraně e grafu G(V, E) přiřazeno reálné číslo w(e) zvané váha, pak G spolu s těmito váhami je nazván ohodnocený graf.*

Nyní si definujeme sousednost uzlů.

Definice 2.4 *Jakékoliv dva uzly, které jsou spojeny hranou, nazýváme sousední uzly.*

Stupeň uzlu v grafu představuje, jak významný daný uzel je. Tato hodnota je rovna počtu uzlů propojených hranou s daným uzlem.

Definice 2.5 *Stupeň uzlu d(v) je počet hran, které s uzlem v sousedí.*

Dále úplný graf.

Definice 2.6 *Úplný graf je takový graf, ve kterém jsou každé dva vrcholy spojené hranou.*

S úplným grafem souvisí trojúhelník.

Definice 2.7 *Trojúhelník je úplný graf se třemi vrcholy.*

Nyní si definujeme sled a cestu. Sled si můžeme představit jako postupné procházení grafem, kde můžeme postupovat mezi uzly pouze pomocí hran. Nic nám nebrání navštívit uzel či cestu i vícekrát.

Definice 2.8 *Sled (z vrcholu u do vrcholu v) v grafu G je libovolná posloupnost* $(u = v_0, v_1, ..., v_k = v)$*, kde* v_i *jsou vrcholy grafu G a pro každé i = 1,...., k je* $v_{i-1}v_i$ *hranou grafu G. Číslo k je délka tohoto sledu. Říkáme, že sled prochází vrcholy* $v_0, ..., v_k$ *nebo že na něm tyto vrcholy leží.*

Cesta je speciální případ sledu, kdy každý vrchol navštívíme pouze 1.

Definice 2.9 *Cesta z u do v v grafu G je sled* $(u = v_0, v_1, ..., v_k = v)$*, kde* v_i*, ve kterém se každý vrchol* v_i *objevuje pouze jednou.*

Dále budeme potřebovat hodnotu grafová vzdálenost. Tato hodnota představuje, jak daleko od sebe uzly jsou v rámci grafu.

Definice 2.10 *Nechť G(V,E) je souvislý graf. Pro vrcholy* v_1, v_2 *definujeme číslo* $d_G(v_1, v_2)$ *jako délku nejkratší cesty z* v_1 *do* v_2 *v grafu G. Číslo* $d_G(v_1, v_2)$ *se nazývá vzdálenost vrcholů* v_1 *a* v_2 *v grafu G.*

D8le si definujeme podgraf. Jedná se o graf složený z libovolného počtu uzlů a hran původního grafu.

Definice 2.11 *Podgraf P grafu G je takový graf, jehož množina uzlů je podmnožinou množiny uzlů grafu G a zároveň množina jeho hran je podmnožinou množiny hran grafu G.*

Další vlastností grafu je souvislost. Tato vlastnost určuje, zda graf tvoří propojený celek nebo je složen z jednotlivých nepropojených částí.

Definice 2.12 *Souvislým grafem se nazývá takový graf, mezi jehož libovolnými dvěma uzly existuje cesta. Silně souvislý je graf právě tehdy, když v něm pro každou dvojici uzlů a,b existuje orientovaná cesta z a do b a nazpět z uzlu b do a.*

Se souvislostí souvisí pojem komponenta. Komponenta je taková část grafu, jenž je souvislá.

Definice 2.13 *Komponentou grafu se nazývá každý jeho maximální souvislý podgraf. Silnou komponentou orientovaného grafu se nazývá každý jeho maximální silně souvislý podgraf.*

2.3 Komunity

Jednou z mnoha zásadních vlastností sítí je rozdělení na komunity. Pojem komunita nemá přesnou definici, ale lze ho popsat takto: Jedná se o takovou skupinu uzlů, kde jsou uzly navzájem silně provázány pomocí hran, avšak se zbytkem hran je provázanost pouze slabá.

Převážně u sociálních sítí jsou komunity důležité, protože to definuje skupiny lidí, kteří mají velmi podobné charakteristiky. V závislosti na zkoumané síti se může jednat o rodinu, či skupinu lidí, kteří mají zájem o stejné zboží. Podle vybraného přístupu může uzel patřit pouze do jedné či i více komunit.

3 Algoritmy vyhledávající komunity

Tato kapitola se věnuje algoritmům, jejichž cílem je nalézt v grafu komunity. Nejsou zde uvedeny veškeré existující algoritmu, jedná se pouze o přehled. Více informací o tomto tématu lze nalézt v článku [6].

3.1 Lokální metody

Jedno ze základních dělení algoritmů pro vyhledávání komunit je rozdělení na algoritmy lokální a globální.

Globální algoritmy jsou algoritmy takové, jenž potřebují znát celou datovou kolekci ještě před spuštěním algoritmu. Tato kolekce se však může dynamicky měnit nebo může být příliš velká na zpracování. Všechny uvedené algoritmy mimo tuto sekci patří mezi globální algoritmy.

Lokální algoritmy jsou algoritmy takové, jenž typicky začnou s jedním uzlem či hranou a pomocí nějaké metriky, či heuristiky postupně přidáváme další uzly a hrany. Tyto algoritmy jsou obzvláště účinné, máme li pouze určit komunitu, do které patří vybraný uzel.

Lokální metody obvykle pracují s pojmy jádro komunity, hranice komunity a pouzdro komunity. Jádro komunity jsou uzly náležící do dané komunity, jenž jsou spojeny hranou pouze s uzly také náležící do stejné komunity. Hranice komunity jsou uzly, jenž patří do komunity a jsou hranou propojeny s uzlem, jenž není součástí komunity. Nakonec pouzdro komunity jsou uzly, jenž jsou hranou propojeny s uzlem tvořícím hranici komunity.
Nyní příklad jedné lokální metody.

3.1.1 Iterative local expansion

Tato metoda posuzuje kvalitu komunity podle ostrosti hranice komunity, neboli aby skupina uzlů byla považována za komunitu, hranice těchto uzlů musí být ostrá. Tato ostrost je počítána jako poměr hran vedoucí z hranice do jádra a počet hran vedoucí do pouzdra.

Algoritmus začne u náhodného uzlu. Při každém kroku je spočítáno, zda při přidání jednoho z uzlů pouzdra do komunity vzroste ostrost. Pokud ostrost vzroste, uzel je přidán do komunity a všechny tři kolekce, tedy jádro,hranice a pouzdro jsou aktualizovány. Tyto kroky opakujeme, dokud neexistuje takový uzel, jenž po přidání do komunity zvýší hodnotu ostrosti.

3.2 Tradiční metody

3.2.1 Metoda dělení grafu

Smyslem této metody je rozdělit vrcholy grafu do několika skupin tak, aby počet hran propojujících tyto skupiny byl minimální. Počet hran mezi těmito skupinami se nazývá velikost řezu. Tato metoda vyžaduje dopředu určit, jak velké skupiny budou a kolik

jich ve výsledku má být. Typickým algoritmem fungujícím na tomto principu je The kernighan-Lin algoritmus

3.2.1.1 The kernighan-Lin algoritmus

Jeden z nejstarších a přitom stále používaný algoritmus využívá dělení grafu. Pro vysvětlení tohoto algoritmu je třeba definovat hodnotu Q, jenž představuje rozdíl mezi počtem hran uvnitř jednotlivých skupin a počtem hran mezi těmito skupinami.

Na začátku algoritmu máme graf rozdělený na dvě části požadované velikosti, ať už rozdělené náhodně nebo pomocí speciální nějaké metody. Poté vybereme stejný počet vrcholů z obou skupin a tyto vrcholy mezi skupinami prohodíme, aby hodnota Q co nejvíce stoupla. Po sérii prohození uzlů, jenž mají za následek snížení i zvýšení hodnoty Q, vybereme takové rozdělení, jenž má nejvyšší Q. Toto rozdělení použijeme pro další iteraci.

Tento algoritmus má docela nízkou časovou náročnost při konstantním počtu prohození za iteraci, a to $\Theta(n^2 \log v)$, kde v je počet vrcholů v grafu. Výsledné rozdělení grafu silně závisí na výchozím rozdělení, jiné algoritmy mohou vracet lepší výsledky. Pro zajištění co nejlepších výsledků, musíme zajistit, aby počáteční rozdělení bylo co nejlepší. Tudíž tato metoda se používá v kombinaci s jinými algoritmy, jenž naleznou vhodné počáteční rozdělení.

3.2.2 Hierarchické algoritmy

Obecně toho předem o struktuře komunit v grafu víme jen velmi málo, tudíž není vhodné předem určovat, kolik komunit se v grafu nachází, či kolik uzlů mají komunity asi obsahovat. V těchto případech nám metoda dělení grafu příliš nepomůže. Hierarchická metoda tyto informace nepotřebuje.

Hierarchické vyhledávání komunit je metoda, která se snaží posupně vytvářet hierarchii komunit. Výsledek této metody je stromová struktura zvaná Dendrogram. Typickou vlastností hierarchické metody je, že nemá ukončení, a pokud nezavedeme ukončovací podmínku, výsledek algoritmu bude nic neříkající.

Silné stránky

- Nemusíme odhadovat počet komunit v grafu.
- Výsledky jsou relativně přesné.

Slabé stránky

- Špatná rychlost.
- Jakmile provedeme nějaký krok, již se nemůžeme vrátit.
- Rozpad velkých komunit na menší celky.
- Slabost vůči šumu.

Hierarchická metoda se dělí na dva typy.

3.2.2.1 Aglomerativní typ Jedná se o takzvaný algoritmus od spodu. Obecně je časová náročnost algoritmu $\Theta(n^3)$, avšak existují rychlejší speciální případy.
Průběh algoritmu je následující.

1. Na začátku algoritmu přiřadíme každému uzlu vlastní komunitu a spočítáme vzdálenosti mezi nimi.
2. Najdeme dvojici komunit s nejkratší vzdáleností a tyto komunity spojíme v jednu.
3. Spočítáme vzdálenosti všech starých komunit s nově vytvořenou komunitou.
4. Opakujeme kroky 2 a 3 dokud nejsou veškeré uzly v jedné komunitě nebo nezasáhne vnější podmínka.

Jak bylo zmíněno, spojujeme komunity s největší blízkostí, ale co to ta blízkost vlastně je? Význam blízkosti se liší podle konkrétně zvoleného metriky.
Nejčastěji mívá následující významy:

- Euklidovská vzdálenost - $\sqrt{\sum_i (a_i - b_i)^2}$
- Manhattan vzdálenost - $\sum_i (a_i - b_i)^2$
- Maximální vzdálenost - Jedná se o největší možnou vzdálenost mezi uzly různých komunit.
- Minimální vzdálenost - Jedná se o nejmenší možnou vzdálenost mezi uzly různých komunit.
- Průměrná vzdálenost - Jedná se o průměrnou vzdálenost mezi uzly různých komunit.

3.2.2.2 Rozdělující typ Jedná se o opačný přístup oproti aglomerativní metodě neboli od shora. Časová náročnost se liší na zvolené metodě výpočtu, rychlejší metody dosahují $\Theta(n^3)$.

Nyní si musíme vyjasnit pojem edge betweenness centrality, dále pouze betweenness. Jedná se o hodnotu, která představuje kolik nejkratších cest vede přes danou hranu. Hrany s vysokou hodnotou betweenness by měly tvořit mosty mezi jednotlivými komunitami. Při odstranění takovýchto hran by měly vzniknout spojité podgrafy reprezentující komunity. Jinými slovy tato hodnota určuje, jak velká je šance, že právě tato hrana odděluje 2 komunity mezi sebou.
Samotný algoritmus probíhá následovně:

1. Na začátku algoritmu přiřadíme každému souvislému podgrafu vlastní komunitu a betweenness pro všechny hrany.
2. Najdeme hranu s nejvyšší hodnotou betweenness a tuto hranu odstraníme.
3. Projdeme graf, zda se nějaký souvislý podgraf nerozdělil na 2 a případně vytvoříme novou komunitu.
4. Opakujeme kroky 2 a 3 dokud nejsou odstraněny veškeré hrany nebo nezasáhne vnější podmínka.

Podobně jako u aglomerativní metody i zde celý algoritmus spočívá ve výpočtu jisté vlastnosti. Tentokrát se jedná o betweenness. O výpočtu této hodnoty je celá tato práce.
Zde je výčet několika příkladů výpočtu betweenness centrality:

- Nejkratší cesty - Kolik nejkratších cest mezi všemi uzly grafu danou hranou prochází.
- Náhodná procházka - Kolikrát je hrana navštívena při náhodném procházení grafu ze všech možných dvojic uzlů.
- Tok v síti - Při převedení grafu na elektrickou síť, jak velký proud by danou hranou tekl.

3.3 Partitional clustering

Další populární metoda vyhledávání komunit v grafech. V této metodě je počet komunit k předem dán. Uzly jsou vloženy do metrického prostoru a mezi dvojicemi uzlů je spočítána vzdálenost. Vzdálenost je dána mírou odlišnosti mezi uzly. Cílem je oddělit uzly do k komunit tak, aby se maximalizovala popř. minimalizovala cena vybrané funkce. Nejčastěji používané funkce jsou:

- Minimum k-clustering - V tomto typu algoritmu má funkce význam průměru komunity, kde průměr komunity je největší vzdálenost mezi dvěma uzly komunity. Tento typ metody se snaží o to, aby průměry jednotlivých komunit byly co nejmenší, tudíž komunity byly "kompaktní".

- K-clustering sum - Podobné jak předchozí typ, průměr je zaměněn za průměrnou vzdálenost mezi veškerými uzly komunity.
- K-center - Pro každou komunitu i je definován referenční bod x_i, centroid, a spočítá se maximální vzdálenost uzlu komunity od těžiště, d_i. Komunity a jejich těžiště jsou určovány tak, aby hodnota d_i byla co nejmenší.
- K-median - Stejné jako k-center, avšak maximální vzdálenost je nahrazena průměrnou vzdáleností.

Nejpoužívanější mezi algoritmy partial clustering je algoritmus algoritmus K-means.

3.3.1 K-Means

Algoritmus založený na přiřazování uzlů nejbližšímu těžišti. Pro větší přesnost může být algoritmus spuštěn vícekrát a výsledky zprůměrovány.
Postup je následovný:

1. Náhodně, či pomocí výpočtu vybereme počáteční uzly a tyto uzly určíme jako jednotlivé centroidy.
2. Pro každý uzel v grafu najdeme nejbližší těžiště a přidáme uzel do této komunity. Težiště komunity přepočítáme.
3. Celý postup opakujeme, kolikrát považujeme zapotřebí a výsledné rozdělení poté zprůměrujeme.

Tento postup je rychlý, ale nebere ohled na pořadí uzlů při zpracování, tudíž může dojít k horším výsledkům. Pomalejší, ale spolehlivější postup v kroku 2 po každé iteraci najde uzel s nejmenší vzdáleností k libovolnému těžišti a zpracuje právě tento.

3.3.2 C-means

Základní myšlenka algoritmu je, že každý uzel může částečně patřit do více komunit s tím že existuje hodnota, jak moc do dané komunity patří. Uzly uvnitř komunity mají tuto hodnotu vyšší než uzly na kraji komunity. Tato hodnota se značí w_k.

Vzorec pro výpočet těžiště komunity je následující:

$$C_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} \tag{1}$$

Celý algoritmus probíhá velmi podobně jako v případě K-Means:

1. Náhodně, či pomocí nějakého úvodního výpočtu vybereme počáteční uzly a tyto uzly určíme jako jendotlivé centroidy.
2. Vypočítáme těžiště pomocí výše uvedeného vzorce pro každou komunitu.

3. Pomocí stejného vzorce vypočítáme pro každý uzel, nakolik patří do které komunity.

4. Kroky 2 a 3 opakujeme, dokud nedojde k ustálení grafu.

3.4 Metody založené na modularitě

Modularita Q, původně sloužící pro účely ukončení hierarchických algoritmů, se rychle rozšířila a stala se významnou součástí mnoha dalších metod. Modularita je z daleka nejznámnější a nejpoužívanější metoda posouzení kvality rozdělení grafu na komunity. Modularita se počítá pomocí vzorce:

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(C_i, C_j) \tag{2}$$

Proměnná m zde představuje celkový počet hran v původním grafu, A_{ij} je rovna jedné, pokud spolu vrcholy i,j sousedí, v jiném případě 0, $k_i k_j$ jsou stupně vrcholů i,j a nakonec $\delta(C_i, C_j)$ má hodnotu 1, pokud jsou uzly i,j ze stejné komunity, jinak má hodnotu 0. Výsledek musí být v rozmezí 0 až 1, kde 0 je nejhorší rozdělení.

Hlavní motivace metod založených na modularitě je, že pokud nalezneme rozdělení s maximální hodnotou Q, nalezli jsme nejlepší rozdělení. Avšak vzhledem k obřímu počtu možností, jak graf rozdělit, je testování veškerých možností rozdělení prakticky téměř nemožné ve vhodném čase. Avšak existují metody, které se pokoušejí alespoň odhadnout vhodné rozdělení.

3.4.1 Hladové metody

Základ jednoho z hladových algoritmů byl již dříve zmíněn. Jedná se o upravenou verzi aglomerativního typu hierarchické metody. Po každém kroku spojíme takové dva uzly, aby hodnota modularity co nejvíce narostla, popř. co nejméně klesla. Po provedení všech možných kroků se podíváme, ve kterém kroku byla hodnota modularity největší. Rozdělení v tomto kroku poté považujeme za výsledek.

Celková doba výpočtu trvá $\Theta((m+n)n)$, kde n je počet vrcholů grafu a m je počet hran.

Nenasytné metody mívají tendenci tvořit velké komunity na úkor komunit malých. Toto může vést k nízkým hodnotám maximální modularity. Kvalitu výsledného rozdělení můžeme zvýšit také, pokud aglomerativní metodu zahájíme s předem spočítanými skupinami uzlů, nikoli samostatnými uzly.

3.4.2 Simulated annealing

Jedná se o metodu založenou na průzkumu jednotlivých možných stavů grafu.Metoda se snaží nalézt globální maximum funkce F. Přechod z jednoho stavu do stavu druhého proběhne s pravděpodobností 1, pokud F po této změně vzroste, jinak s pravděpodobností

$\exp(\beta\delta F)$, kde δF je pokles hodnoty F a β je index stochaického šumu, jenž se po každé iteraci zvyšuje. Tato hodnota zajišťuje, aby se algoritmus nezastavil v lokálním maximu.

Po nějaké době se algoritmus dostane do stabilního stavu, jenž považujeme za výsledek. Tato metoda se může velice přiblížit skutečné maximální hodnotě modularity, avšak tato metoda je velmi pomalá. Časová náročnost nemůže být odhadnuta, jelikož průběh silně závisí na parametrech a ne pouze velikosti grafu.

3.4.3 Modifikace modularity

V poslední době se začaly objevovat modifikace a rozšíření modularity. Tyto změny jsou obvykle zapříčiněny specifickými algoritmy, či speciálními typy grafů.

Například modularita může být snadno rozšířena pro práci s ohodnocenými grafy. Ve vzorci výpočtu modularity jednoduše zaměníme stupně uzlů k_i a k_j za sílu stupňů s_i, s_i, jenž jsou rovny součtu hodnot hran náležících uzlu. Dále je třeba počet hran m nahradit součtem hodnot všech uzlů W. Výsledný vzorec tedy odpovídá:

$$Q = \frac{1}{2W}\sum_{ij}(W_{ij} - \frac{s_i s_j}{2W})\delta(C_i, C_j) \tag{3}$$

Dalším případem rozšíření modularity může být modularita pro orientované grafy. V takovém případě je vzorec:

$$Q = \frac{1}{m}\sum_{ij}(A_{ij} - \frac{k_i^{out} k_j^{in}}{m})\delta(C_i, C_j) \tag{4}$$

V tomto vzorci jsme použili místo stupňů uzlů počty vstupních a výstupních hran $k_i^{out}k_j^{in}$.Nakonec kombinace obou případů, tedy orientovaný ohodnocený graf, má vzorec:

$$Q = \frac{1}{W}\sum_{ij}(W_{ij} - \frac{s_i^{out} s_j^{in}}{W})\delta(C_i, C_j) \tag{5}$$

3.5 Alternativní metody

V této sekci se budeme věnovat dalším algoritmům, jenž nešly zařadit do předešlých kategorií. K jistým podobnostem s předchozímy algoritmy však může dojít.

3.5.1 Šíření značek

Jedná se o rychlou a jednoduchou metodu. Uzlům je na začátku přiřazena unikátní značka. Během každé iterace se v náhodném pořadí každý uzel převezme značku, jenž má nejvíce sousedů. Pokud žádná značka nemá většinu, vybere se náhodně z nejčastějších značek sousedů. Většina značek časem zmizí, avšak pár značek získá převahu. Algoritmus končí, jakmile dojde k vyváženému stavu, tedy každý uzel má stejnou značku, jako většina jeho sousedů. Rozdělení do komunit poté odpovídá uzlům se stejnýma značkama.

Díky této konstrukci zajistíme, aby každý uzel měl více sousedů ve komunitě, ke které náleží, než v komunitách ostatních. Podle testů na skutečných grafech, i když tato metoda obsahuje náhodný element, výsledky jsou si většinou velmi podobné.

Výhodou této metody je, že nepotřebuje žádné informace o tom, jak má být graf rozdělen, tedy např. počet komunit. Rychlost jedné iterace je $\Theta(m)$, kde m je počet uzlů. Kolik iterací je potřeba provést se zdá být nezávislé na velikosti grafu, tudíž je tato metoda velmi rychlá.

3.5.2 L-shell

Jedná se o algoritmus, jenž pro vybraný uzel určí, do jaké komunity patří. Komunity jsou definovány lokálně, na základě jednoduchého principu počtu hran uvnitř a vně skupiny uzlů. Postup algoritmu je následující. Začneme v zadaném uzlu a postupně přidáváme uzly ležící na dalších vrstvách. Vrstva je skupina uzlů, jenž se nachází v určité vzdálenosti od původního uzlu. V každé iteraci tedy procházíme čím dál vzdálenější uzly. Během každé iterace počítáme, kolik je hran jdoucích z nové vrstvy vně doposud nalezený celek, a kolik dovnitř. Pokud poměr těchto dvou čísel překoná předem danou hodnotu, jsou uzly této nové vrstvy přidané do komunity a proces se opakuje. Pokud tohoto poměru nedosáhneme, algoritmus končí.

3.5.3 Bridge Bounding

Metoda podobná L-shell. Během této metody však dochází k růstu komunity, dokud nenarazíme na některou z hraničních hran. Tyto hrany můžeme rozpoznat různými způsoby, například hodnotou betweenness či edge clustering coeficient (obě tyto hodnoty jsou vysvětleny v další kapitole). Problém je, že často není jasný rozdíl mezi vypočítanou hodnotou hraniční a nehraniční hrany, tudíž musíme nastavit konstantní hodnotu, odkdy se hrana pokládá za hraniční.

4 Betweenness hran

V minulé kapitole jsme se dozvěděli o existenci rozdělující hierarchické metodě vyhledávání komunit. Jak již bylo zmíněno, samotný algoritmus je jednoduchý, pouze postupně odstraňujeme hrany s nejvyšší hodnotou betweenness, dokud nejsme spokojeni s výsledkem. Nyní však nastává problém. Jak vlastně tuto hodnotu spočítat? Jak již bylo uvedeno v kapitole 3, v sekci rozdělující typ, betweenness je hodnota určující, jak velká je šance, že tato hrana odděluje 2 komunity.
Nyní uvedu způsoby výpočtu, jenž byly v rámci práce nalezeny a naimplementovány:

4.1 Metoda nejkratší cesty

Jedná se o nejpoužívanější algoritmus, z jehož základu vychází spousta upravených verzí. Hlavní myšlenka algoritmu spočívá v nalezení hran, které oddělují velké husté podgrafy mezi sebou. Logicky přes takové hrany v případě, že pro každou dvojici zkusíme najít nejkratší cestu, povede největší množství těchto cest. V případě, že bychom vyhledávali cestu pro každou dvojici uzlů zvlášť, časová náročnost by byla příliš vysoká.

4.1.1 Neohodnocený graf

V případě, že graf nemá ohodnocené hrany, s řešením, jak nejkratší cesty spočítat s přijatelnou časovou složitostí, přisel Newman [7]. Tato metoda spočívá v tom, že pro jeden uzel spočítáme nejkratší cesty ke všem zbývajícím uzlům. Toho dosáhneme díky prohledávání do šířky. V případě, že ke každému uzlu existuje pouze jediná nejkratší cesta, bude tento výpočet pro jeden uzel trvat $\Theta(m)$, kde m je počet hran v grafu. Nyní máme strom nejkratších cest a jak postupně procházíme strom od listů ke kořenu, přidáváme hranám betweenness o hodnotě 1 + množství uzlů pod touto hranou. Celková doba spočítání betweenness trvá $\Theta(nm)$, kde n je počet uzlů a m je počet hran v grafu. Tento výpočet musíme provést pro každý uzel jako počáteční, čili celková časová náročnost je $\Theta(n^2m)$

V reálných sítích však existuje více nejkratších cest mezi dvěma uzly, než pouze jedna. Proto vznikla varianta výpočtu, kde pokud pro dvojici uzlů existuje více nejkratších cest, pak se hodnota betweenness pro hrany na těchto cestách musí rovnoměrně rozdělit tím, že betweenness vydělíme počtem cest.

Nejúčinnější pro vyhledávání nejkratších cest je vyhledáváním do šířky. Během prohledávání sousedů, veškeré jeho dříve nezpracované sousedy dáme do fronty, a totéž provedeme se sousedy těchto uzlů, dokud neprojdeme veškeré uzly grafu.
Celý algoritmus tedy probíhá takto:

1. Počátečnímu uzlu přidělíme vzdálenost 0. Počáteční uzel umístíme do fronty pro nenavštívené uzly.

2. Vybereme první uzel z fronty a označíme ho jako aktuální. U každého uzlu sousedícímu s aktuálním uzlem, který ještě nebyl navštíven, porovnáme vzdálenost.

Pokud je tato vzdálenost větší než současná vzdálenost, vzdálenost i aktuálnš nejkratší cestu nahradíme současnými hodnotami. Pokud jsou stejné, vzdálenost zachováme, ale uložíme další cestu. Pokud je menší, neděláme nic.

3. Krok 2 opakujeme, dokud se fronta nevyprázdní.

4. U každého uzlu spočítáme, kolik cest jsme k tomuto uzlu nalezli a každé hraně na těchto cestách přiřadíme betweenees vydělená počtem cest.

Tato metoda je jedna z nejrychlejších a přesto vrací dobré výsledky. Metoda funguje pouze s minimální úpravou jak pro orientované, tak neorientované grafy.

4.1.2 Ohodnocený graf

Pro ohodnocený graf výše uvedená metoda nebude fungovat. Princip vyhledávání do šířky spočívá v tom, že nám záleží pouze na počtu hran, kterými cesta prochází. Čím méně je hran na cestě, tím je cesta kratší. V případě ohodnoceného grafu však závisí i na hodnotách hran, ne jen na jejich počtu. Tudíž by mohl nastat případ, kde hrana s hodnotou 100 je považována za kratší cestu, než dvě hrany s hodnotami 5. S řešením tohoto problému přišel již v roce 1959 Edsger Dijkstra [8].

Dijkstrův algoritmus, podobně jako v předchozím případě, nalezne pro jeden uzel nejkratší cesty do všech ostatních uzlů v grafu.
Postup je následující:

1. Veškerým uzlům v grafu přiřadíme maximální možnou vzdálenost, počátečnímu uzlu vzdálenost 0.

2. Nastavíme veškerým uzlům vlastnost nenavštíveno. Vytvoříme množinu uzlů obsahující všechny dosud nenavštívené uzly.

3. Aktuálnímu uzlu spočítáme vzdálenosti k ještě nenavštíveným sousedům jako vzdálenost do aktuálního uzlu + hodnota hrany vedoucí k sousednímu uzlu. Porovnáme tyto hodnoty s hodnotami vzdálenosti, již uzly již obsahují. Pokud je tato hodnota menší, hodnotu přepíšeme a zaznamenáme, kudy jsme k tomuto uzlu přišli.

4. Jakmile projdeme všechny sousedy aktuálního uzlu, označíme uzel jako navštívený a odstraníme ho z množiny nenavštívených uzlů. Tento uzel již nebude navštíven.

5. Vybereme jeden z uzlů z kolekce nenavštívených uzlů takový, aby jeho vzdálenost od kořene byla co nejmenší. Tento uzel nastavíme jako aktuální a vracíme se ke kroku 3.

6. Pokud je kolekce nenavštívených uzlů prázdná, právě jsme dokončili algoritmus pro nalezení všech nejkratších cest z jednoho uzlu.

Výše zmíněný postup musíme opakovat tolikrát, kolik je v grafu uzlů. Časová náročnost oproti neohodnocenému grafu je vyšší, protože potřebujeme pro každý krok projít množinu nenavštívených uzlů a najít, který je právě nejbližší uzel od kořene. Časová náročnost je tedy $\Theta(m+n^2)$, kde m je počet hran a n je počet uzlů. Tento algoritmus musíme provézt pro každý uzel, tedy celková časová náročnost je $\Theta(m + n^3)$.

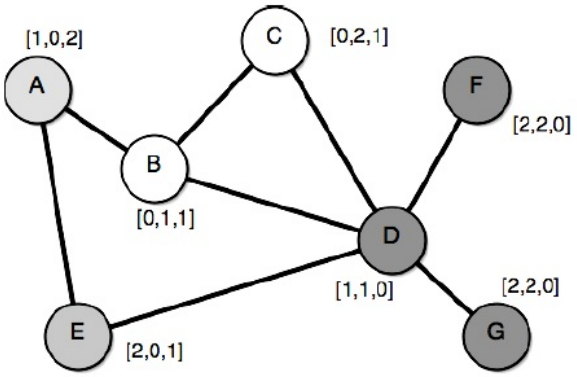
4.1.3 Distance to zone

Další variantou algoritmu využívající nejkratších cest je algoritmus uvedený v této publikaci [9]. Jedná se o algoritmus, jenž se na rozdíl od algoritmů předchozích, nesnaží vypočítat přesné hodnoty betweenness, ale pouze tyto hodnoty odhadnout. Základní princip algoritmu spočívá v rozdělení grafu na zóny a dimenze. Zóna představuje pomyslné rozdělení grafu na jednotlivé oblasti, zatímco dimenze odpovídá jednomu konkrétnímu rozdělení grafu do zón. Tento algoritmus funguje pouze pro souvislé grafy.
Způsob tvorby zón probíhá takto:

1. Náhodně vybereme uzly, které budou představovat jádra zón.
2. Projdeme sousedy všech uzlů, jež jsou již přiřazeny do zón, a tyto uzly vložíme do seznamu.
3. Ze seznamu uzlů vybereme náhodně uzel a přiřadíme ho do zóny svého souseda, v případě sousedů s různými zónami, rozhodneme náhodně.
4. Aktualizujeme seznamy uzlů přiřazených do zón a seznam jejich sousedů. Vracíme se ke kroku 3, dokud nepřiřadíme všechny uzly do zón.
5. Tímto máme vytvořenou první dimenzi. Tuto dimenzi si uložíme a vracíme se ke kroku 1, dokud nevytvoříme požadovaný počet dimenzí.

Množství zón a dimenzí silně ovlivňuje dobu potřebnou ke zpracování dat. Toto množství však také zvyšuje kvalitu odhadu hodnot betweenness. Tímto máme připravené dimenze a zóny. Nyní však potřebujeme spočítat vzdálenost všech uzlů od zón pro všechny dimenze, a tyto hodnoty uložit.

1. Vybereme, pro kterou zónu a dimenzi budeme vzdálenosti počítat. Všem uzlům v této zóně nastavíme vzdálenost 0. Tyto uzly si označíme jako vyřazené a umístíme do seznamu aktuálních uzlů.
2. Najdeme všechny sousedy všech aktuálních uzlů a přiřadíme jim vzdálenost aktuálních uzlů +1.
3. Všechny tyto nově nalezené uzly nastavíme jako vyřazené a list aktuálních uzlů změníme na tyto uzly.
4. Opakujeme od kroku 2, dokud nevyřadíme všechny uzly v grafu.
5. Jakmile ukončíme výpočet vzdáleností, přesuneme se na další zónu popř. na celou další dimenzi. Ve výsledku by si měl každý uzel zapamatovat, ve které dimenzi patří do jaké zóny. Dále by si měl pamatovat, ve které dimenzi jsou vzdálené jednotlivé zóny od tohoto uzlu.



Obrázek 1: 3 různé zóny a jejich vzdálenosti. Prvni číslo je vzdálenost do bílé zóny, druhé číslo do šedé zóny, třetí číslo do tmavé zóny. Čerpáno z [9]

Nakonec musíme spočítat samotnou hodnotu beetweenness. Metoda, pojmenovaná jako DTZ (Distance to zone), spočívá ve vyhledávání nejkratších cest ne mezi dvojicemi uzlů, ale mezi uzlem a všemi zónami. Vzhledem k tomu, že se jedná pouze o odhad, použijeme pouze asi 50 procent hran. Počet hran je samozřejmě možné změnit pro vyšší přesnost. Díky předešlému výpočtu vzdáleností zón je nalezení nejkratší cesty primitivní. Na obrázku 1 můžeme vidět, jak vypadá ohodnocení DTZ pro jednu dimenzi.

1. Náhodně vybereme uzel.
2. Vybereme, ke které zóně se snažíme nalézt cestu. Vybraný uzel označíme jako aktuální.
3. Zkontrolujeme veškeré sousedy aktuálního uzlu. Vybere z nich takový, u kterého jsme dříve vypočítali nejkratší cestu k potřebné zóně. Hraně mezi uzly zvýšíme hodnotu betweenness a uzel označíme jako aktuální.
4. Opakujeme od kroku 3, dokud nenalezneme uzel se vzdáleností 0 (uzel patřící do hledané zóny).
5. Opakujeme od kroku 2, dokud nenalezneme cesty pro všechny zóny, pro všechny dimenze.
6. Opakujeme od kroku 1, dokud nepoužijeme požadované množství jedinečných uzlů.

Tento algoritmus má však problém, že pokud jsou uzly v jedné zóně ve více dimenzích, je jen malá šance, že i kdyby se jednalo o významné uzly, odhadovaná hodnota betweenness bude nízká. Tento problém se řeší vyšší hodnotou dimenzí, pomocí následující úpravy. Jakmile najdeme cestu k zóně a nalezneme uzel patřící do zóny, spočítáme běžnou metodou nejkratší cesty mezi tímto uzlem a všemi uzly v této zóně. Oproti běžnému způsobu však ve výsledku může být hodnota betweenness pro každou hranu navýšena pouze o jedna, aby se zamezilo vysokému vzrůstu v okolí počátečního uzlu. Vzhledem k tomu, že se jedná o algoritmus, který pouze odhaduje betweenness a ve kterém hrají velkou roli náhodně vybrané uzly a počáteční zóny, můžou se výsledky při každém spuštění lišit v závislosti na množství zón dimenzí a počtu testovaných uzlů. Čím větší hodnoty do těchto proměnných dosadíme, tím se budou výsledky méně lišit a budou přesnější.

4.1.4 Ostatní

Existují ještě další úpravy jednotlivých algoritmů. Zde budou v rychlosti uvedené 2 tyto metody, které však pro svou jednoduchost a malou výpovědní hodnotu, nebyly implementovány. Více informací můžete například najít v článku [7].

První metoda je odhad hodnot betweenness pomocí vzorku uzlů. Cíl této metody je zrychlit algoritmus za cenu přesnosti. Principem metody je, že nevyhledáváme nejkratší cesty mezi veškerými dvojicemi uzlů, ale pouze vybraným vzorkem dvojic uzlů. Podle velikosti vzorku můžeme ovlivnit rychlost a kvalitu výpočtu.

Druhá metoda omezuje, přes kolik uzlů může náhodná cesta vést. Ačkoli se tímto výpočet mírně urychlí, hlavní smysl tohoto omezení je, že cesty vedoucí přes velké množství uzlů ztrácí původní význam vazeb.

4.2 Metoda náhodné procházky

Jedná se asi o výrazně nejpomalejší z používaných algoritmů pro výpočet betweenness. Tomuto algoritmu se mimo jiné věnuje tento článek [10]. Hodnota betweenness u tohoto algoritmu vypovídá, kolikrát průměrná náhodná procházka mezi dvěma uzly projde danou hranou. Samozřejmě opět postupně pro veškeré dvojice uzlů.

Tento algoritmus pracuje pouze s grafy tvořené maticí sousednosti a tyto grafy musí být spojité. Postup je následující:

1. Vytvoříme matici A. Tato matice představuje matici sousednosti.
2. Vytvoříme matici D. Tato matice představuje matici stupňů. Jinými slovy na diagonále matice je číslo odpovídající počtu sousedů daného uzlu.
3. Vytvoříme matici M. Tato matice představuje D-A.
4. Nyní si určíme, ze kterého uzlu začínáme procházky. Z matice M odstraníme řádek a sloupec odpovídající tomuto uzlu, poté matici invertujeme a nakonec vrátíme vynulovaný řádek na původní pozici.
5. Nyní máme připravenou matici, díky které jednoduše vyčteme, jaké jsou pro jednotlivé hrany hodnoty betweenness. Čteme však pouze řádky a sloupce představující počáteční uzel.
6. Opakujeme od kroku 3, dokud nepoužijeme jako výchozí uzel všechny uzly.

Z popisu výše vyplývá, že matice o velikosti počtu uzlů musí být invertována, navíc ne jednou ale tolikrát, kolik máme uzlů. Díky tomuto je algoritmus oproti jiným metodám velmi pomalý. Rychlost algoritmu odpovídá $\Theta((n+m)m+n^2)$, kde n je počet uzlů v grafu, a m je počet hran. Záleží však na zvoleném algoritmu pro inverzi matice.

4.3 Radicchi algoritmus

Tento algoritmus oproti ostatním uvedeným algoritmům nepočítá přímo hodnotu betweenness. Tento algoritmus počítá hodnotu edge-clustering coefficient. Tato hodnota představuje, v kolika trojúhelnících z možných se tato hrana nachází. Oproti betweenness je význam této hodnoty opačný, tedy místo vyřazení hrany s nejvyšší hodnotou, vyřadíme hranu s hodnotou nejnižší. Oproti betweenness se tato hodnota počítá pouze lokálně, díky tomu se jedná o velmi rychlou metodu.

Tato metoda se snaží brát v potaz, že hrany uvnitř komunity budou silně provázány a oproti tomu uzly náležící hranám mezi komunitami budou mít jen malé množství sousedů. Vzorec pro výpočet hodnoty edge-clustering coefficient je následující:

$$C_{i,j} = \frac{z_{i,j}+1}{min[(k_i-1),(k_j-1)]} \tag{6}$$

Čitatel, tedy $z_{i,j}$, představuje množství trojúhelníků, do kolika tato hrana patří. Hodnota +1 slouží, aby hrany s nízkým počtem trojúhelníků nebyly trestány příliš tvrdě.

Jmenovatel, tedy $min[(k_i - 1), (k_j - 1)]$, představuje maximální množství trojúhelníků, kterých tato hrana mohla být součástí. Neboli stupeň hraničního uzlu s nižším stupněm, snížený o hodnotu 1. jak vyplývá z předešlé věty, k_i a k_j jsou tedy stupně uzlů i,j.

Existuje více variant této metody, avšak jediný rozdíl je pouze v tom či hledáme trojúhelníky, čtyřúhelníky až n-úhelníky. V práci byly implementovány nejpoužívanější verze, a to verze s trojúhelníky a čtyřúhelníky.

Tato metoda je používaná pro svou rychlost, avšak v grafech, kde se nachází pouze malý počet trojúhelníků, je tato metoda značně nepřesná. Rychlost metody se odhaduje na $\Theta(m^2 + m)$, kde m je počet hran v grafu.

Více se o tomto algoritmu se můžete dočíst například zde [11].

5 Implementace

5.1 Jednotlivé metody

V této sekci bude popsána implementace jednotlivých metod výpočtu betweenness a popis problémů, které během implementace vznikly. Nebude popisován obecný postup algoritmů, či jejich princip. Veškeré rychlostní testy byly provedeny na počítačové sestavě s procesorem AMD Athlon(tm) II X4 631 Quad-Core 2.60GHz, 4GB ram.

5.1.1 Nejkratší cesta

Jedná se o nejpoužívanější metodu, díky vysoké rychlosti a slušné kvalitě výsledků. Vstupem implementace je list obsahující aktuální podgrafy. Na konci metody bude každá hrana všech podgrafů ohodnocena vypočítanou hodnotou betweenness, pomocí které můžeme později určit, kterou hranu odstranit. Tato metoda existuje v různých variantách podle typu grafu, či způsobu výpočtu.

- První dělení je podle typu grafu. Podle toho zda se jedná o orientovaný či neorientovaný graf, musíme algoritmus náležitě upravit. Nejedná se o velkou změnu, pouze při hledání sousedních uzlů musíme brát v potaz, že hrany jsou orientované.
- Druhé dělení je podle stylu výpočtu. Můžeme nalézt jednu nejkratší cestu mezi uzly, nebo budeme hledat, zda neexistuje více stejně dlouhých nejkratších cest, mezi jejichž hrany hodnotu betweenness rozdělíme. Úprava algoritmu z předešlého souvětí jasně vyplývá. Místo vyřazení hrany okamžitě po nalezení cesty musíme dál sledovat cesty vedoucí mezi těmito uzly, zda nenajdeme stejně dlouhou jinou variantu. Tímto se zhorší rychlost algoritmu, avšak výsledky jsou přesnější.
- Poslední dělení metody nejkratší cesty je, zda graf má ohodnocené hrany, či nikoli. Oproti předešlým dělením, zde je nutné použít naprosto jiný algoritmus, nestačí pouhá úprava stávajícího. Jak bylo popsáno v sekci Betweenness, pro ohodnocené grafy je nutné použít Dijskrův algoritmus, který je pomalejší, než algoritmus základní.

Kombinací všech těchto dělení vzniklo 8 metod, avšak orientované verze metod nebyly plánovány a tudíž ani testovány. Nyní přejdeme k naměřeným rychlostem. Pro porovnání jednotlivých metod byl náhodně vygenerován graf obsahující 1000 uzlů a 1500 hran. Dále, pro zajištění objektivního měření, bude měřen pouze jeden průběh výpočtu betweenness, ne celého času potřebného pro vyhledání celého rozdělení uzlů do komunit, což by bylo časově náročné, navíc by bylo měření zkresleno různými podružnými funkcemi, jako vyřazení hran, či rozpoznávání jednotlivých podgrafů. naměřené rychlosti jsou v tabulce 1.

Algoritmus:	1 cesta	Více cest	Ori.1 cesta	Ori.více cest
Rychlost:	1,33s	2,3s	9,8s	11,1s

Tabulka 1: Rychlost nejkratší cesty pro graf velikosti 1000 uzlů a 1500 hran

- Verze algoritmu, která nalezne pouze jednu nejkratší cestu pro tuto datovou kolekci trvá 1,33 sekund.
- Verze algoritmu, která nalezne všechny nejkratší cesty pro tuto datovou kolekci trvá 2,3 sekund. Oproti jednoduché varianty je to výrazný nárůst. Při dalším testování však bylo zjištěno, že tento nárůst roste jen pozvolna. Pro graf o velikosti 2000 uzlů a 2500 hran byl tento rozdíl pouze 2 sekundy, což vzhledem k celkovému nárůstu doby pro větší graf na 8 sekund je nárůst zanedbatelný.
- Orientovaná verze vyhledávající pouze jednu nejkratší cestu trvá 9,8 sec. Oproti předešlým algoritmům se jedná o výrazný nárůst. Mimo jiné je tento algoritmus pomalejší díky nutnosti po úplně každém použití uzlu nalézt z ostatních uzlů ten s nejmenší aktuální vzdáleností.
- Orientovaná verze vyhledávající veškeré nejkratší cesty trvá 11,1 sekund. Rozdíl oproti jednodušší metodě probíhá stejně jako v případě neorientovaných grafů.

5.1.2 Radicchi

Tato metoda by měla být oproti jiným metodám daleko rychlejší, ale její použití je silně limitováno, jelikož grafy neobsahující velké množství trojúhelníků (popř. n-úhelníků podle varianty) budou vracet nepřesné výsledky. Porovnání výsledků rozdělení uzlů do komunit bude uvedeno v poslední kapitole a to kapitole věnující se testování. Celý algoritmus je velmi jednoduchý a v praxi pouze musíme spočítat, do kolika n-úhelníků, podle varianty, hrana patří. Oproti jiným metodám však nakonec výsledky musíme invertovat nebo musíme odstranit hranu s nejmenší, a ne největší hodnotou. Naimplementovány na ukázku byly varianty s trojúhelníky a čtverci. Pro porovnání rychlostí bude použit stejný graf, jako při testování rychlosti nejkratší cesty. Pro další posouzení rychlosti byla použita ještě druhá generovaná kolekce obsahující 3000 uzlů a 7000 hran. Naměřené výsledky jsou v tabulce 2.

Algoritmus:	Trojúhelníky	Čtyřúhelníky
Rychlost kolekce1:	0,007s	0,02s
Rychlost kolekce2:	0,12s	0,21s

Tabulka 2: Rychlost Radicchi pro grafy o velikosti 1000/1500 a 3000/7000

Podle výsledků je jasné, že tento algoritmus je skutečně velmi rychlý.

5.1.3 Distance to zone

Jedná se o experimentální implementaci, jelikož původní autoři této metody již nepopsali, jak vlastně hodnotu betweenness získat. Tato metoda má proměnné jako počet použitých uzlů, počet vytvořených zón a počet vytvořených dimenzí. Podle těchto hodnot se bude přesnost výsledků a rychlost lišit. Pro testovaní rychlosti, i výsledného rozdělení bylo vybráno následující nastavení: 3 dimenze, počet zón odpovídá 20% počtu uzlů a vyhledávat cesty budeme pomocí 50% uzlů v grafu.

Algoritmus se dělí na 2 samostatné části. V první části se snažíme graf rozdělit do jednotlivých zón a poté spočítat vzdálenosti veškerých uzlů ke všem zónám. Tato část je relativně časově náročná, avšak pro každou 1 vyřazenou hranu stačí být provedena jednou.

Druhá část algoritmu je již počítání nejkratších cest. Byly zvoleny dvě metody, jak betweenness počítat. První metoda náhodně vybere daný počet uzlů, a nalezne nejkratší cesty mezi těmito uzly a jednotlivými zónami. Poté jen spočítáme, kolikrát přes každou hranu vedla nějaká cesta. Druhá metoda na první navazuje, a poté co cesta dorazí k zóně, dále spočíta nejkratší cesty od tohoto uzlu ke všem uzlům dané zóny. Toto vše je provedeno pro veškeré existující dimenze. Pro měření byla pouze použita menší z dříve použitých datových kolekcí, tedy graf obsahující 1000 uzlů a 1500 hran. Naměřené údaje v tabulce 3

Algoritmus:	Pouze cesty k zónám	Včetně uvnitř zón
Rychlost:	1,23s	6,55s

Tabulka 3: Rychlost DTZ pro graf velikosti 1000 uzlů a 1500 hran

- Verze algoritmu hledající pouze cesty k zónám trvá 1,23 sekund. Oproti metodě nejkratší cesty, kterou to má urychlit je ušetřeno 1,1 sekund.
- Verze algoritmu hledající cesty i uvnitř zón trvá 6.55 sekund. Jak můžeme vidět, při zvoleném nastavení tato metoda trvá dokonce 3 krát déle než původní metoda nejkratší cesty.

Zatím se zdá, že metoda hledající cesty i uvnitř zón je příliš časově náročná. Ovšem také záleží, jak bude rozdělení komunit přesné.

5.1.4 Náhodná procházka

Princip této metody byl již uveden v sekci betweenness. Oproti ostatním algoritmům je tato metoda výpočtu založená na práci s maticemi. Rychlost této metody silně závisí na zvoleném inverzním algoritmu, avšak všeobecně se má jednat o velice pomalou metodu. Vzhledem k pomalosti této metody byl vytvořen speciální graf o velikosti 200 uzlů a 400 hran. Na tomto grafu trval jeden krok 18 sekund. Více jak polovinu této doby zabralo samotné invertování. Můžeme videt, že tento algoritmus je velice pomalý, a proto v praxi téměř nepoužívaný.

5.1.5 Porovnání

Ze všech algoritmů je z daleka nejrychlejší Radicchi a to v obou variantách. Vděčí svoji rychlosti především lokálnímu počítání, oproti ostatním globálním metodám. Tento algoritmus by bez problému zvládal vyhledat komunity i v grafech s několika tisíci uzlů a hran. Výsledky této metody jsou však nejisté, a budou dále posouzeny v další kapitole.

Další rychlý algoritmus je Distance to zone - základní verze. Tento algoritmus je však pouze aproximační, takže jeho výsledky nemusí vůbec odpovídat realitě. Dále mezi relativně rychlé algoritmy můžeme zařadit klasický, ověřený a nejčastěji používaný algoritmus nejkratší cesty - neohodnocený. Jeho varianty nám umožní výběr mezi rychlostí a přesností. Rozdíl rychlosti mezi variantami se sice pohybuje pouze kolem 1-2 sekund, avšak pokud bude třeba provést velké množství kroků pro nalezení komunit, jedná se již o několik minut. Vzhledem k všeobecnému rozšíření verze rozpoznávající více cest, byl tento algoritmus zvolen jako referenční během testování rozdělení do komunit. Veškeré tyto algoritmu lze doporučit pro grafy do 600 ulzů a 800 hran, záleží, co považujeme za přijatelnou dobu a jaké ukončovací podmínky nastavíme.

Zdaleka nejhůře si vedl algoritmus náhodné cesty, jenž i na daleko menší datové kolekci měl velice špatný čas a lze jej doporučit maximálně pro velmi malé grafy velikosti asi 150 uzlů a 250 hran.

5.2 Popis ukončovacích podmínek

V předešlých kapitolách již několikrát bylo zmíněno, že hierarchické algoritmy mají nejasné zakončení, a pokud je necháme bez speciální ukončovací podmínky, postupně z grafu odstraní veškeré hrany. Takový výsledek by však nic neznamenal.

Žádné oficiální ukončovací podmínky však neexistují. Můžeme si zkusit na internetu nějaké podmínky najít, nebo můžeme sami zkusit nějaké vytvořit. Pro tuto práci byly použity následující podmínky.

5.2.1 Počet odstraněných hran

Tato podmínka je ze všech podmínek nejjednodušší a nejsnáze člověka napadne. Jak název napovídá, jednoduše cyklus ukončíme, jakmile je z grafu odstraněn daný počet hran. Tato podmínka je vhodná především pro měření rychlostí, či porovnávání metod. V praxi však můžeme jen hádat, kolik kroků,tedy odstraněných hran, musí být provedeno, aby výsledek odpovídal skutečnému rozdělení uzlů do komunit. Implementace této podmínky je naprosto jednoduchá, tj. počítadlo kroků a podmínka. Použitím této podmínky nemusí dojít k nalezení žádných nových komunit, pokud je graf velmi hustý a počet kroků nastaven na nízké číslo.

5.2.2 Počet nalezených komunit

Opět jedna z lehčích podmínek. Existuje celá řada algoritmů, která dokonce vynucuje dopředu odhadnout, kolik komunit asi graf obsahuje. Jedná se opět o lehkou podmínku ve smyslu implementace. Vzhledem k tomu, že po každém kroku stejně zjišťujeme, zda

vznikly nové komunity, moc nového se dělat nemusí. Opět ale nám tato podmínka nedá nijak najevo, jak dobré rozdělení je, a pouze provede rozdělení do daného počtu komunit, i když se v grafu reálně nachází méně komunit, než se snažíme vytvořit. Tímto může vzniknout zbytečné oddělování jednotlivých uzlů, či jejich dvojic od komunity jen proto, aby byl naplněn požadovaný počet.

5.2.3 Požadovaná hustota komunit

Jedná se již o něco těžší podmínku, avšak stále docela jednoduchou. Princip této podmínky spočívá v tom, že podle popisu komunit, by uzly v rámci jedné komunity měly být nadprůměrně provázány. Oproti ostatním metodám, tato podmínka postupně vylučuje komunity, se kterými již dále nebudeme pracovat, jelikož je již považujeme za skutečné výsledné komunity. Samotný postup je tedy následující.

Po rozdělení komunity na dvě, tyto nově vzniklé komunity vždy otestujeme speciální metodou, jenž posoudí, zda jsou tyto komunity považovány za husté. Aby byla komunita považována za hustou, musí obsahovat přednastavený počet hran. Tento počet se může lišit, v této implementaci se jedná o 50% všech možných hran, které by v této komunitě byly, jednalo by se o úplný podgraf. Pokud program považuje komunitu za hustou, tak se již dále s touto komunitou nepracuje. V moment, kdy jsou všechny komunity považovány za husté, algoritmus je ukončen.

5.2.4 Modularita

Modularita je hodnota, jak moc kvalitní je rozdělení uzlů do komunit. Modularita slouží pouze jako odhad, jak asi dobré toto rozdělení je. Existuje více metod, jak modularitu počítat. Pro implementaci byla zvolena modularita vycházející z článku Newmana [7], viz rovnice 2.

Stejně jako podmínka požadované hustoty i tato podmínka místo pevně daných kritérií využívá spíše odhadu kvality rozdělení grafu. Vzhledem k náročnosti výpočtu modularity a nutnosti počítat modularitu po odstranění každé hrany, tato podmínka zpomaluje celkovou dobu potřebnou k nalezení rozdělení komunit.

Modularita vyjde jako čísla velikosti 0 až 1, kde 1 je nejkvalitnější rozdělení. Samotné číslo nám však nepomůže ukončit algoritmus ve vhodný moment. Mohli bychom sice zavést minimální hodnotu, při jejimž dosažení algoritmus ukončíme, avšak tato minimální hodnota by se příliš graf od grafu lišila. Proto se většinou používá tzv."peak modularita".

Peak modularita je modularita taková, že během dalších n kroků modularita již nestoupá. Tudíž pokud nalezneme rozdělení s vyšší modularitou, než jsme aktuálně považovali za maximální, toto hodnocení se uloží a poté se provede dalších n kroků. Pokud již vyšší modularitu nenalezneme, vracíme se k uloženému rozdělení a algoritmus ukončíme.

5.3 F-Score

Cílem tohoto projektu nebylo posoudit rychlost jednotlivých metod, i když i tyto údaje byly letmo prozkoumány, nýbrž posoudit kvalitu rozdělení uzlů do komunit a vyzkoušet experimentální metody výpočtu. Avšak pro posouzení kvality rozdělení uzlů potřebujeme nějakou metodu porovnání. V aplikaci byl implementován způsob sloužící k porovnání jednotlivých komunit mezi testovaným rozdělením uzlů a rozdělením referenčním. V sekci testování bylo jako referenční rozdělení použito rozdělení pomocí metody nejkratší cesty vyhledávající i více nejkratších cest, avšak to neznamená, že se musíme omezit právě na tuto referenční metodu.

Rozdělení se porovnává stylem, že pro každou komunitu z testovaného rozdělení vypočítáme, jak vysokou hodnotu tzv. F-Score má pro jednotlivé komunity z referenčního rozdělení. Poté komunitu s nejvyšší hodnotou F-Score k této testované komunitě přiřadíme. Jedna referenční komunita může být obrazem více testovaných, avšak pro každou testovanou komunitu existuje pouze jedna referenční.

Než se dostaneme k samotnému výpočtu F-Score, musíme si definovat několik proměnných a jak tyto proměnné vypočítat.

- TP (True positive) - Jedná se o počet vrcholů testované komunity, jenž se nachází i v komunitě referenční.
- FP (False positive) - Jedná se o počet vrcholů testované komunity, jenž se nenachází v referenční komunitě.
- FN (True negative) - Jedná se o počet vrcholů referenční komunity, jenž se nenachází v testované komunitě.

Nyní potřebujeme vypočítat tyto dva údaje:

- P (Precision) - $P = \frac{TP}{TP+FP}$
- R (Recall) - $R = \frac{TP}{TP+FN}$

Tímto jsme získali veškeré potřebné údaje k vypočítání F-Score. Vzorec F-Score je následující: $F = 2 * \frac{P*R}{P+R}$.

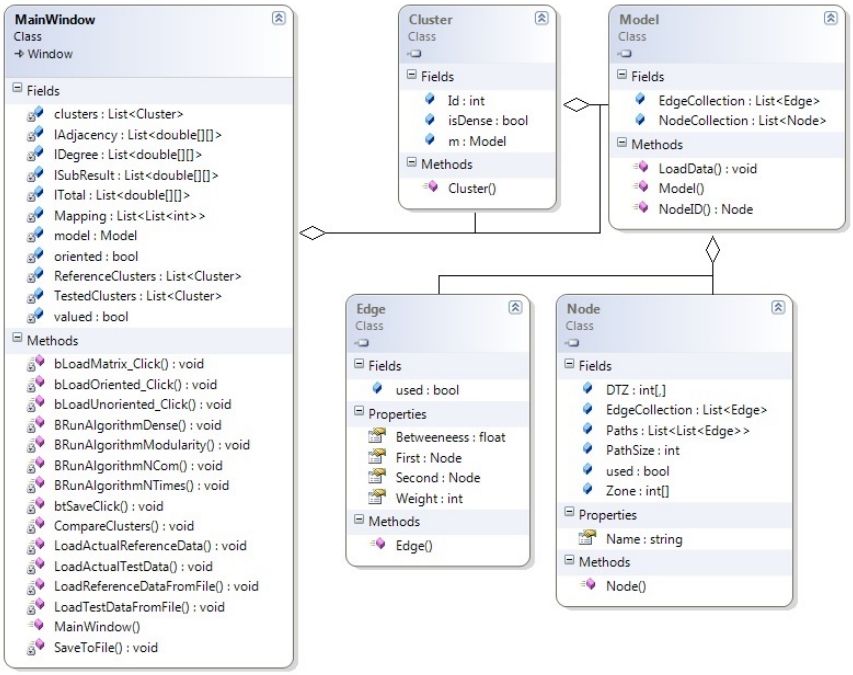
Nyní už jsen stačí projít výsledky F-Score pro všechny referenční komunity a vybrat komunitu, která má tuto hodnotu nejvyšší a tyto dvě komunity porovnat.

5.4 Diagramy

Nyní přejdeme ke způsobu, jakým byla aplikace jako celek implementována. Projekt byl naprogramován v prostředí Visual studio 2010, v programovacím jazyku c#. Pro uživatelské rozhraní bylo použito WPF.

5.4.1 Třídní diagram hlavní části

Na obrázku 2 můžeme vidět základní třídy aplikace. Třída MainWindow slouží jako roz-



Obrázek 2: Třídní diagram

hraní, které podle podmětů uživatele patřičně reaguje a volá metody ostatních, většinou statických tříd. Dále si tato třída pamatuje, jaký typ grafu byl načten a samotný graf, ať už v podobě seznamů, či matic.

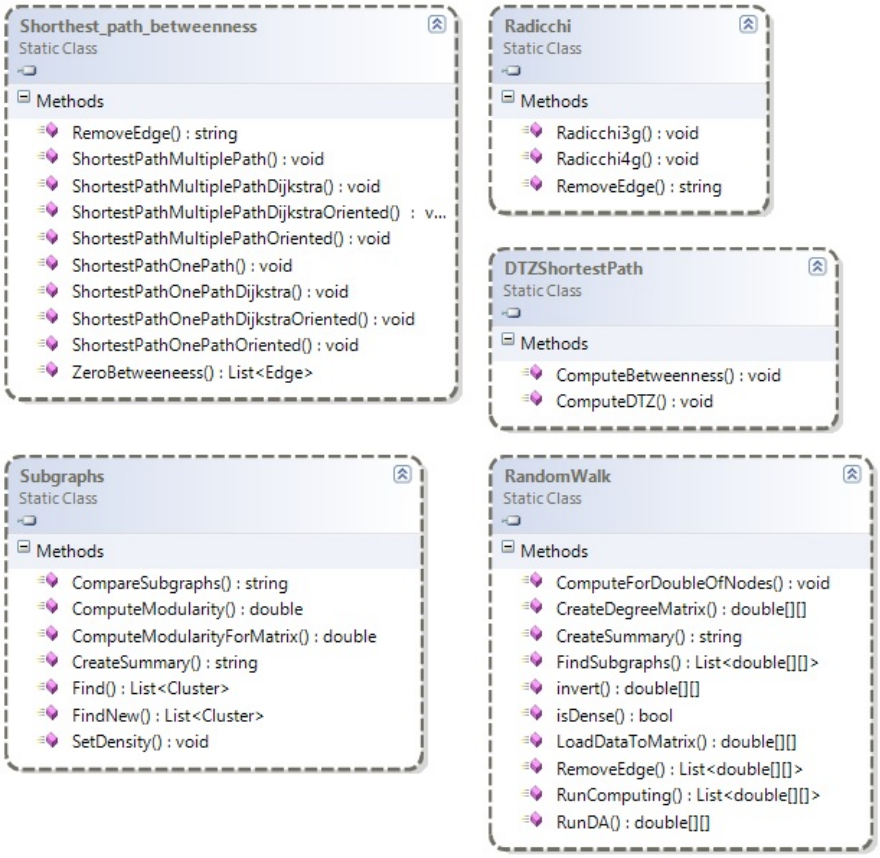
Třída model obsahuje pouze seznamy uzlů a hran tvořené třídami Node a Edge a funkce sloužící pro nahrání dat a nalezení konkrétního uzlu podle jeho id. Původně algoritmy počítající betweenness požadovaly na vstup právě model, později však v rámci zobecnění přístupu a potřebě některých z metod pracovat pouze se spojitými podgrafy, byly vstupy všech metod nahrazeny třídou Cluster.

Třída Cluster reprezentuje jeden konkrétní spojitý podgraf. Podgraf obsahuje své id, seznamy uzlů a hran uložené přes instanci třídy Model a nakonec proměnou isDense, která slouží pouze při použití ukončovací podmínky dle hustoty.

Nakonec třídy Node a Edge, které reprezentují jednotlivé uzly a hrany. Edge obsahuje typické údaje o hranách, jako dvojici uzlů náležící k hraně, její hodnotu, jedná-li se o vážený graf a nakonec hodnotu betweenness. Node obsahuje kromě vlastního id, větší množství proměnných potřebných pro různé metody výpočtu betweenness.

5.4.2 Třídní diagram statických tříd

Tím jsme si prošli jádro aplikace. Nyní si projdeme jednotlivé statické třídy, které většinou slouží k výpočtu betweenness a podružným funkcím kolem této činnosti.



Obrázek 3: Třídní diagram statických tříd

Na obrázku 3 můžeme vidět čtyři třídy pro výpočet betweenness a třídu Subgraphs věnující se funkcím okolo podgrafů a případně jiným podružným operacím. Začneme právě touto třídou.

Třída Subgraphs, jako všechny zde uvedené třídy, je statická. To znamená, že nemusíme vytvářet instance třídy, abychom měli přístup k metodám této třídy. Primární funkce této třídy je z modelu obsahující seznamy uzlů a hran vytvořit seznam spojitých podgrafů pomocí třídy Cluster. Tuto činnost provedeme pouze jednou a to při načítání dat. Během

jednotlivých cyklů, kdy se odstraňují hrany, pro ušetření času vyhledáváme podgrafy již pouze v instancích třídy Cluster, ve kterých se nacházela naposledy odstraněná hrana. Tímto zajistíme značné úspory času.

Dále tato třída obsahuje metodu počítající hustotu clusteru, jenž se používá opět pouze u ukončovací podmínky pomocí hustoty. Tato třída také generuje textový výpis podle momentálního seznamu instancí třídy Cluster a nakonec může sloužit k porovnání rozdělení uzlů mezi námi zvoleným referenčním rozdělení a testovaným rozdělení.

Ostatní statické třídy slouží pouze k výpočtu betweenness a případnému odstranění hran. Třída DTZ nemá vlastní odstranění hrany, jelikož se jedná o stejný postup jako u nejkratší cesty.

Nakonec zmíním třídu RandomWalk. Jelikož se jedná o jediný algoritmus využívající matice, musely zde být vytvořeny speciální metody jako načtení dat do matice. Tato metoda se navíc dělí na množství menších celků, jenž se používají během výpočtu postupně.

5.4.3 Sekvenční diagram

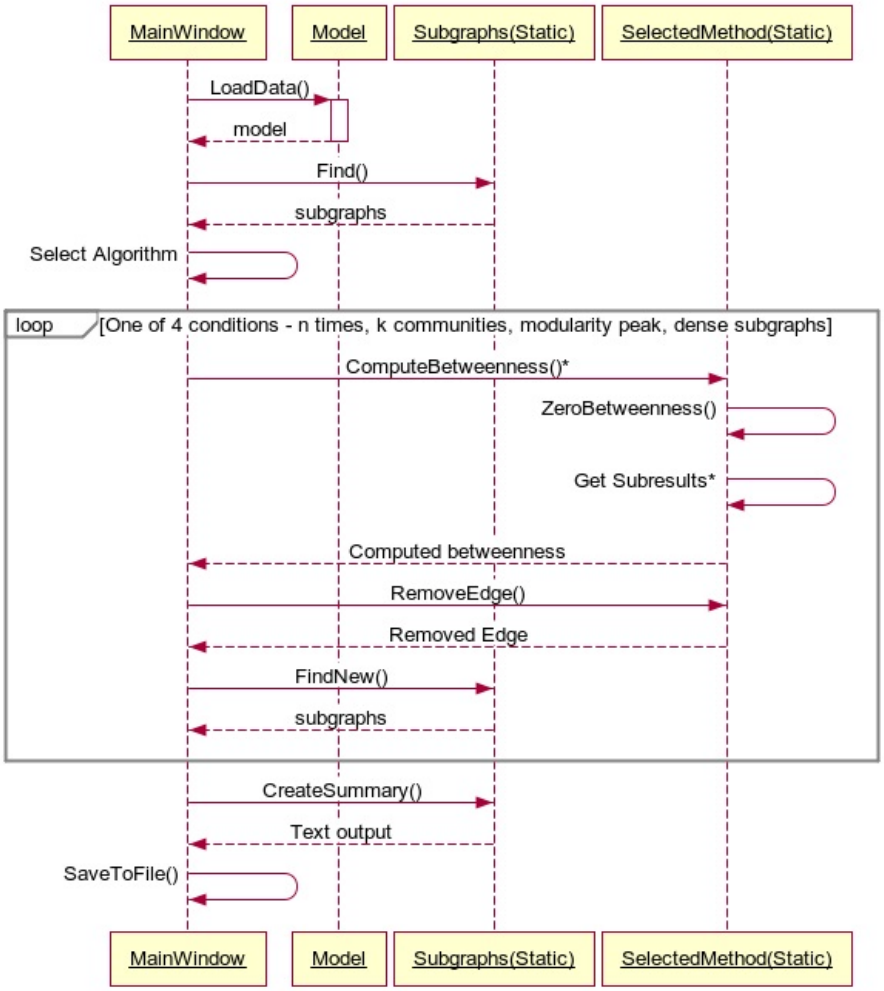
Nyní si pomocí sekvenčního diagramu projdeme, jak vypadá průběh aplikace z hlediska programu. Jak můžeme vidět na obrázku Sekvenční diagram, skutečný běh aplikace začíná v moment, kdy uživatel nechá načíst data. Jakmile jsou data načtena, použijeme statickou knihovnu Subgraphs pro nalezení jednotlivých podgrafů v původním grafu.

Nyní musí uživatel zvolit, jaká metoda výpočtu má být použita a jakou ukončovací podmínku hodlá použít. Nyní se dostaneme do cyklu, který bude probíhat, dokud nedojde k naplnění ukončovací podmínky.

Uvnitř cyklu začneme počítáním betweenness zvolenou metodou. Nejprve však potřebujeme vynulovat jakékoli zbytky dat z předešlého výpočtu. Nyní provedeme všechny potřebné mezi výpočty, až nakonec vrátíme graf s vypočtenými hodnotami betweenness. Nyní je potřeba odstranit hranu. Vzhledem k metodě Radicchi a náhodné procházce, musíme i metodu k odstranění hrany volat z příslušné statické třídy. Nakonec využijeme metody ze statické třídy Subgraphs k získání nové sady podgrafů, pokud nějaké vznikly.

Jakmile je cyklus ukončen, zbývá již pouze nechat vytvořit textový výstup a případně tento výstup uložit do souboru. Uživatel má dále možnost porovnat výsledky s dřívě provedenými pokusy.

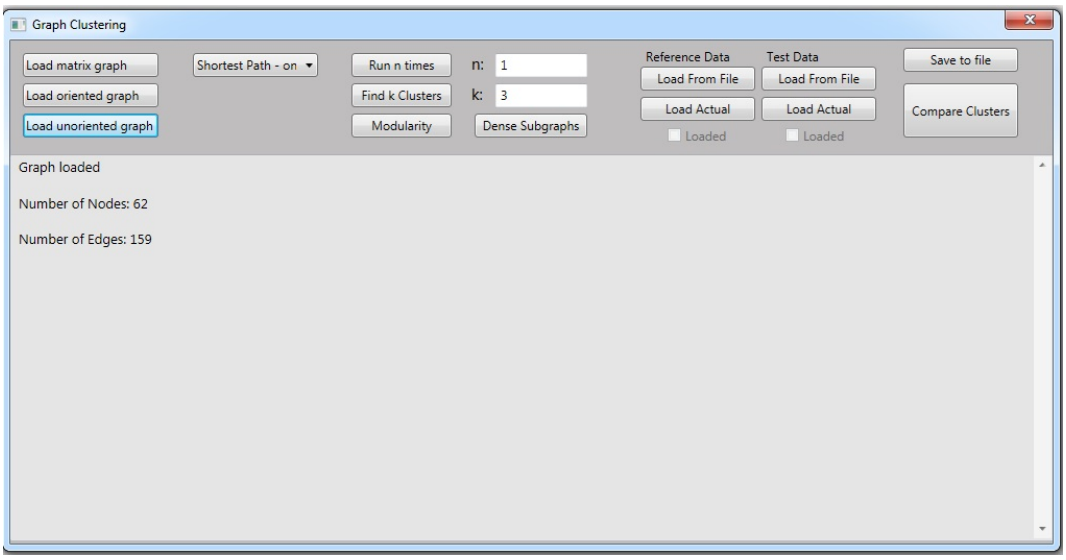
Tímto byl popsán průběh hlavní část programu z pohledu aplikace. V další sekci popíšeme práci s rozhraním a pohled na činnost z hlediska uživatele.



Obrázek 4: Sekvenční diagram

5.5 Práce s aplikací

Práce s programem se dá dělit na 2 samostatné celky. První celek, jehož ovládání se nachází především v levé části obrazovky, je samotné počítání betweenness a rozdělování do komunit. Na pravé straně se nachází část, jenž nám umožní porovnat aktuálně, či dříve vytvořené rozdělení uzlů do komunit.



Obrázek 5: Uživatelské rozhraní

5.5.1 Rozdělení komunit

Ještě než začneme, popíšeme si formát vstupů a výstupů aplikace. Jako vstup slouží libovolný textový soubor, jehož obsahem je graf ve formátu seznamu hran, kde jedna hrana je právě na jednom řádku zapsána jako $Uzel_1; Uzel_2$. Výstup, jenž bude vypsán ve spodní části programu je ve tvaru $NazevKomunity : Uzel_1; Uzel_2; ..; Uzel_3$. nyní k samotnému postupu.

Nejprve je třeba si zvolit, jaký typ grafu chceme nahrát. Program automaticky rozhodne, zda se jedná o ohodnocený graf, či nikoliv. Musíme však dát najevo, zda chceme graf v maticovém tvaru, a zda je graf orientovaný. Nejprve tedy pomocí jednoho ze tří tlačítek v levém rohu načteme zvolený graf. Pokud všechno proběhlo správně, měla by se naplnit nabídka výběru metod. Tato nabídka obsahuje pouze ty metody, které je možné použít pro načtený graf.

Nyní je třeba vybrat, jakou ukončovací podmínku chceme použít. Vrchní tlačítko provede algoritmus n krát, n zadáme do pole vedle tlačítka. Střední tlačítko bude provádět algoritmus, dokud nenalezne k komunit, k se opět zadává do pole vedle tlačítka. Spodní tlačítko spustí algoritmus s podmínkou vrcholu modularity. Vedle tohoto tlačítka se nachází tlačítko pro poslední možnou ukončovací podmínku a to hustota komunit.

Jakmile je ukončovací podmínka splněna, běh programu se zastaví a do textového pole dole se vypíše výsledné rozdělení. V případě, že chceme tento výsledek uložit do souboru, v pravém horním rohu se nachází tlačítko umožňující ukládání výstupu do souboru. V případě, že se nepodaří podmínku splnit, či je splněna již před začátkem algoritmu, jsme o tom v části obsahující výstup informováni.

5.5.2 Porovnání rozdělení

Ovládání této části funkcionality je velmi snadné a skládá se pouze z pěti tlačítek. Nejprve načteme referenční rozdělení. Pokud chceme nahrát tyto data ze souboru, zvolíme vrchní levé tlačítko. Máme však i možnost použít aktuální výstup programu, jenž nahrajeme spodním levým tlačítkem. Pokud vše proběhlo v pořádku, objeví se pod tlačítky potvrzení. Obdobně vybereme testované rozdělení.

Nakonec spustím porovnání pomocí většího tlačítka napravo. Výstup se opět zobrazí v spodní části obrazovky. Výstup je delší, ale jeho nejdůležitější část se nachází dole.

6 Experimenty

V této kapitole se pokusíme vyhodnotit, jak kvalitně jednotlivé algoritmy rozdělí uzly do komunit. Pro posouzení kvality rozdělení byla zvolena hodnota F-Score. Jak se tato hodnota počítá a co znamená jsme si již probrali v dřívější kapitole, nyní zbývá pouze vybrat referenční rozdělení.

6.1 Okolnosti testů

Jako referenční rozdělení bylo vybráno rozdělení generované pomocí algoritmu vyhledávání nejkratší cesty, hledající i více cest mezi dvěma uzly. Tato metoda byla vybrána, protože je všeobecně pokládána za standardní metodu se spolehlivými výsledky.

Jako vstupní data byla zvolena reálná datová kolekce s názvem Dolphins obsahující 62 uzlů a 159 hran. Jedná se o sociální síť delfínů skákavých, žijících v Doubtful Sound na Novém Zélandu. Tato síť byla vytvořena sedmiletým pozorováním interakce delfínů. Tato síť byla také použita a dokonce graficky znázorněna v článku [10]. Druhá testovaná datová kolekce je graf lesmiserables obsahující 77 uzlů a 254 hran. Tato kolekce je hranově ohodnocená, avšak pro potřeby testu byly hodnoty hran odstraněny.

Pro přehlednost testů byla zvolena ukončovací podmínka, jenž ukončí algoritmus, jakmile bude graf rozdělen na 8 komunit.

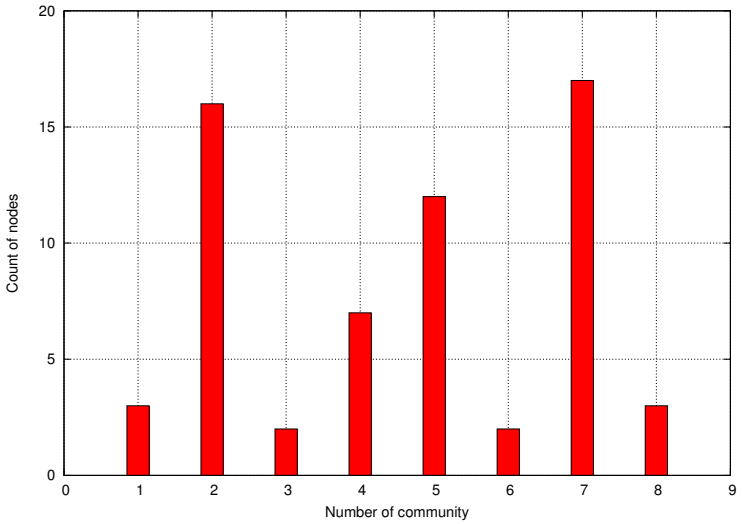
6.2 Testovaná pomocí kolekce dolphins

6.2.1 Referenční rozdělení

Nyní pro možnost porovnání bude uvedena tabulka s grafem zobrazující referenční rozdělení, tedy rozděleníuzlů do komunit, jaké v následujících testech budeme považovat za správné.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	3	16	2	7	12	2	17	3

Tabulka 4: Referenční rozdělení



Obrázek 6: Referenční rozdělení

Z tabulky 4 a grafu na obrázku 6 můžeme vidět, že graf byl rozdělen na 4 velké komunity a 4 malé.

Další podkapitoly se budou věnovat výsledkům F-Score pro jednotlivé algoritmy. Bude vždy uvedeno jaké nejvyšší F-Score nalezená komunita dosáhla, ke které z původních komunit z referenčního grafu je nejvíce podobná a bude uveden počet uzlů těchto nových komunit.

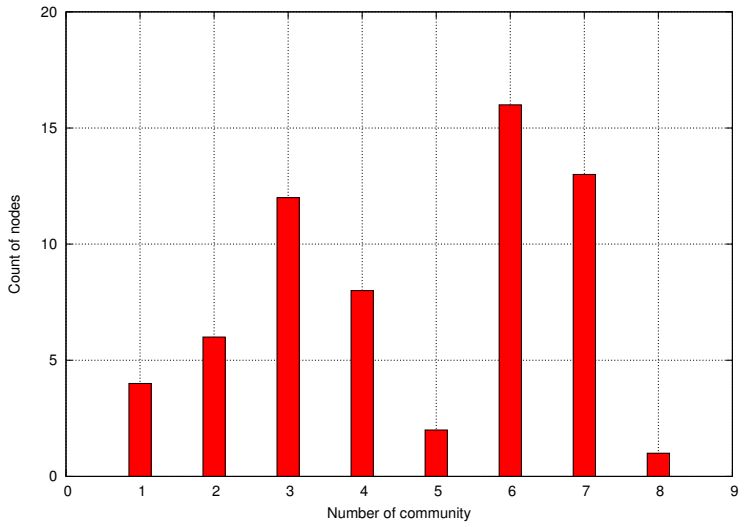
6.2.2 Nejkratší cesta - jedna cesta

Začneme algoritmem, jenž je pouze úpravou referenčního algoritmu. Jediný rozdíl je, že hledáme mezi dvojicemi uzlů pouze jednu nejkratší cestu a po její nalezení se již dále o tuto dvojici nestaráme. Zjištěné hodnoty jsou v tabulce 5.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	4	6	12	8	2	16	13	1
Max. F-Score:	0.86	0.55	1	0.8	1	0.97	0.69	0.11
Odpovídající komunita:	6	1	3	4	5	2	7	8

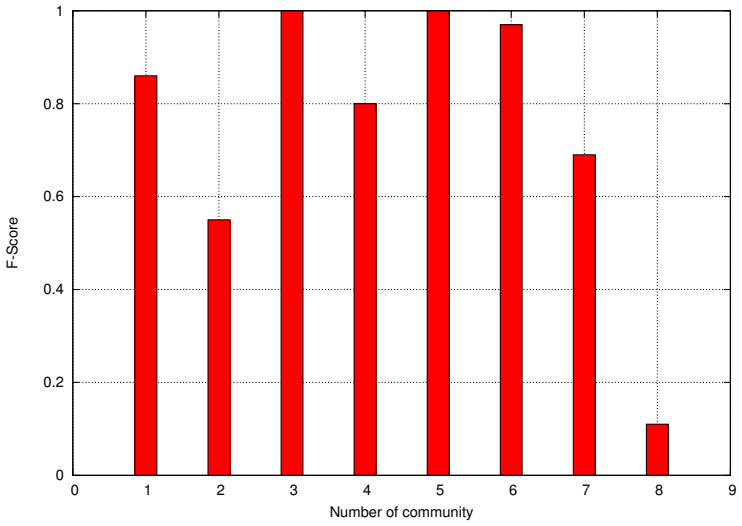
Tabulka 5: Nejkratší cesta - jedna cesta

První řádek tabulky je id komunit nalezených pomocí testovaného algoritmu. Druhý řádek je počet uzlů, jenž tato komunita obsahuje. Třetí řádek je, jakou nejvyšší hodnotu F-Score v porovnání s referenčními komunitami, se podařilo naleznout. Nakonec poslední řádek určí, která komunita z referenčního grafu odpovídá této hodnotě, tedy je nejpodobnější nově nalezené.



Obrázek 7: Nejkratší cesta - počty uzlů

V grafu na obrázku 7, tedy grafu představující počty uzlů, vidíme, že počty uzlů jsou mezi komunitami rozděleny podobně, jako v referenčním rozdělení, akorát jedna z větších komunit se rozdělila mezi menší komunity.



Obrázek 8: Nejkratší cesta - F-Score

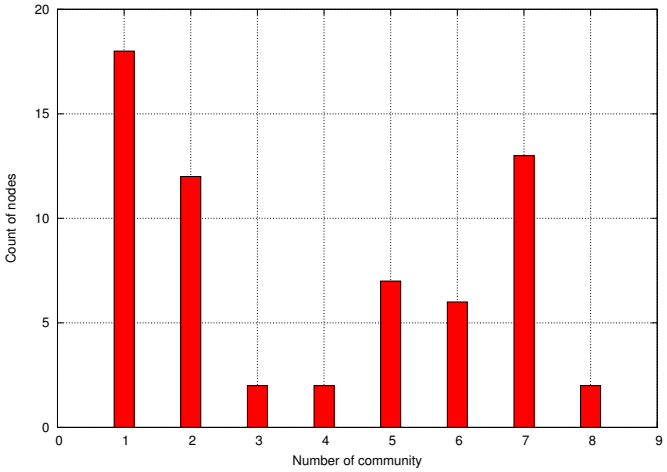
Podle grafu na obrázku 8, tedy grafu představující F-Score, je vidět, že hodnoty F-Score se většinou blíží naprosté shodě, tedy hodnotě 1. Jediná komunita, která nedosáhla F-Score ani 0,5, je komunita 8, avšak díky své zanedbatelné velikosti jednoho uzlu, se tato hodnota nedá považovat za přílišnou odlišnost.
Máme-li vyhodnotit výsledky oproti referenčnímu algoritmu, jedná se o výsledky velice podobné. Není se moc čemu divit, když se jedná o variantu totožného algoritmu. Jak bylo zmíněno u popisu algoritmu, tato metoda by měla dosahovat lepších časů a vzhledem k dobrým výsledkům F-Score, může být tato metoda použita, pokud potřebujeme ušetřit alespoň trochu času.

6.2.3 Náhodná procházka

Náhodná procházka je algoritmus, jenž potřebuje pro svůj výpočet velmi dlouhou dobu. Tato doba je tak vysoká, že se tento algoritmus v praxi téměř ani nepoužívá. Nyní si alespoň porovnáme kvalitu výsledků. Vypočtené hodnoty jsou v tabulce 6.

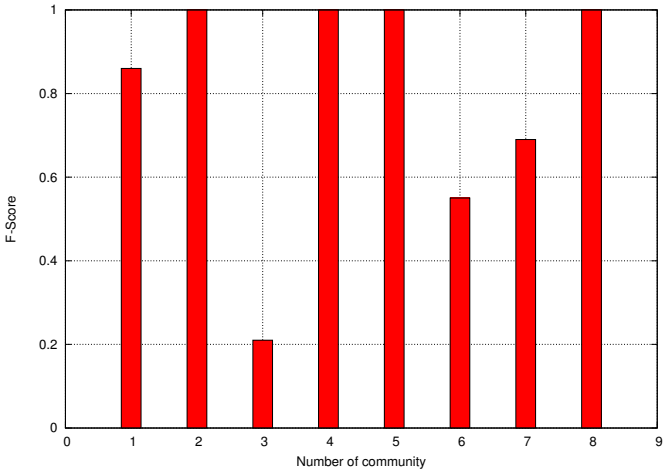
Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	18	12	2	2	7	6	13	2
Max. F-Score:	0.86	1	0.21	1	1	0.55	0.69	1
Odpovídající komunita:	3	4	5	6	1	7	2	8

Tabulka 6: Náhodná procházka



Obrázek 9: Náhodná procházka - počty uzlů

Podle grafu na obrázku 9 byl graf opět rozdělen na 3 velké komunity a několik menších. Můžeme tedy předpokládat, že rozdělení proběhlo podobně jako při použití referenčního algoritmu.



Obrázek 10: Náhodná procházka - F-Score

Podle předpokladu jsou hodnoty F-Score na obrázku 10 skutečně velice vysoké, dokonce ve čtyřech komunitách došlo k naprosté shodě. Zásadní rozdíl se objevil v komunitě 3, jenž představuje referenční komunitu 5. Pokud tyto komunity porovnáme, zjistíme, že tato komunita byla zásadně zmenšena, a její uzly přerozděleny mezi ostatní menší komunity. Díky tomu byly hodnoty některých ostatních komunit také sníženy.
Metoda náhodné procházky sice vrací dobré výsledky, avšak díky velice nízké rychlosti je tato metoda téměř nepoužívána. Není důvod používat takto pomalou metodu, když máme k dispozici jiné metody, např. metodu nejkratší cesty, jenž vrací podobné výsledky v daleko lepších časech.

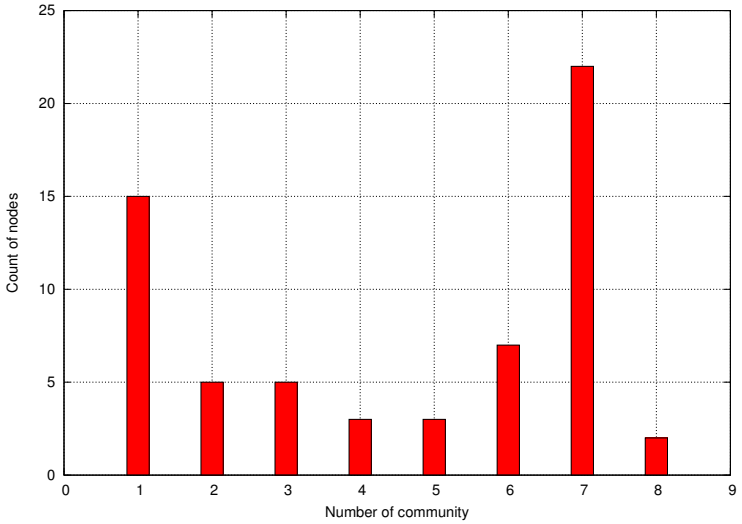
6.2.4 Radicchi - trojúhelníky

Nyní přejdeme k porovnání výsledků metody Radicchi, ve které jsme hledali trojúhelníky. Tato metoda se hlavně vyznačuje svoji extrémně velkou rychlostí. Tato metoda však také může vracet velice špatné výsledky u grafů s nízkým počtem trojúhelníků. Vypočtené hodnoty v tabulce 7.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	15	5	5	3	3	7	22	2
Max. F-Score:	0.59	0.86	0.45	0.62	1	0.48	0.32	1
Odpovídající komunita:	2	7	5	3	6	4	1	8

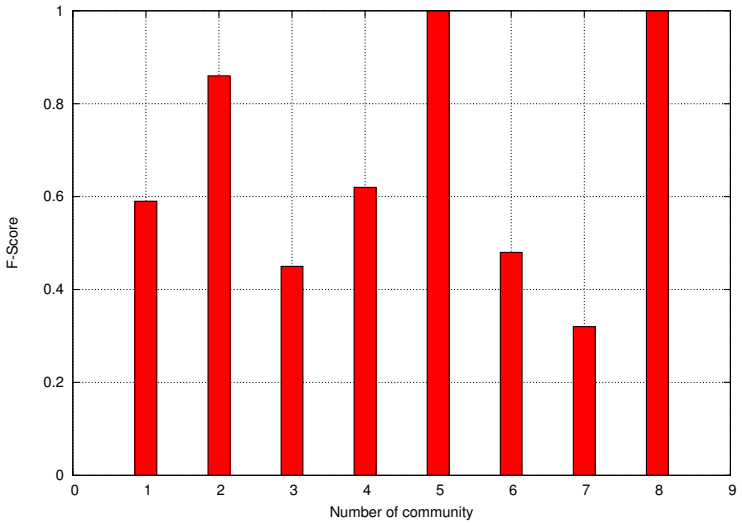
Tabulka 7: Radicchi - trojúhelníky

Podle záznamů v tabulce 7 se zdá, že zmizely některé z větších komunit. Podíváme se na grafy, kde půjdou výsledky lépe porovnat.



Obrázek 11: Radicchi, trojúhelníky - počty uzlů

Skutečně vymizely některé z větších komunit za cenu nárustu velikosti komunity 7. Ilustrace na obrázku 11.Nyní se skusíme podívat, jak tato skutečnost ovlivní výsledky F-Score.



Obrázek 12: Radicchi, trojúhelníky - F-Score

I když graf na obrázku 12 obsahuje dvakrát naprostou shodu, jedná se o shodu pouze miniaturních komunit. Většina komunit mají průměrné hodnoty F-Score. Komunita 7, jenž obsahuje většinu uzlů grafu, má hodnotu F-Score pouze 0.32, což je velmi nízká hodnota.
Výsledek této metody je oproti referenčnímu rozdělení docela odlišný. Tato metoda však má správně fungovat, jak už bylo několikrát zmíněno, pouze u grafů obsahující velké množství trojúhelníků. Je tedy možné, že testovací data nebyla pro tuto metodu vhodná. I přes nepřesné výsledky může být tato použita například, pokud potřebujeme nalézt komunity v opravdu velkém grafu, nebo jsme si jistí, že náš graf splňuje požadavek pro použití této metody.

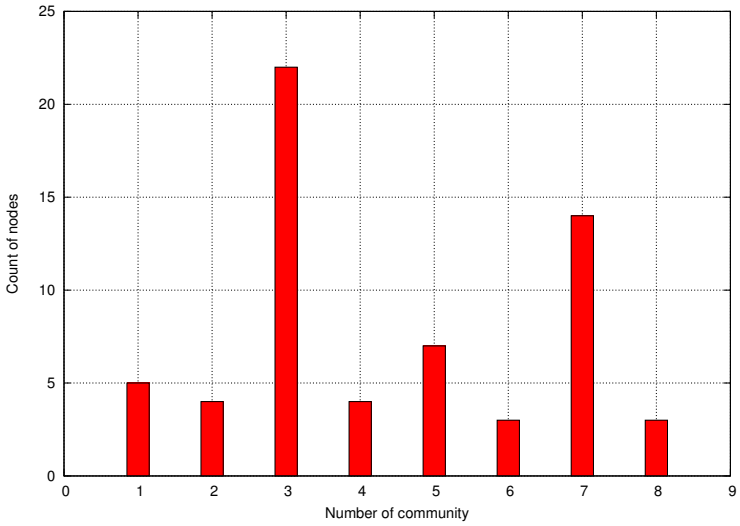
6.2.5 Radicchi - čtyřúhelníky

Tato metoda je pouze upravená verze předešlé metody. Zkusíme se tedy podívat, zda vyhledávání čtyřúhelníků namísto trojúhelníků zlepší výsledky. Vypočtená data jsou v tabulce 8.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	5	4	22	4	7	3	14	3
Max. F-Score:	0.86	0.83	0.44	0.84	0.92	0.29	0.3	0.3
Odpovídající komunita:	4	3	1	5	2	6	7	8

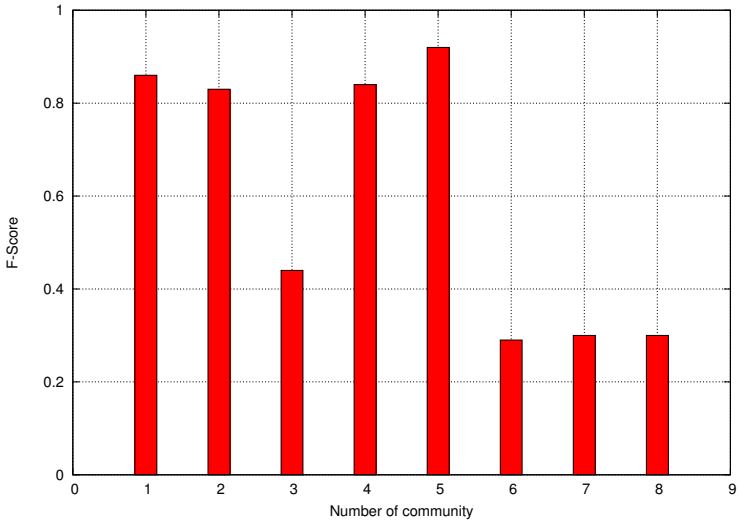
Tabulka 8: Radicchi - čtyřúhelníky

Hodnoty v tabulce 8 nevypadáji oproti předešlému případu příliš odlišně. Jak tedy vypadají grafy?



Obrázek 13: Radicchi, čtyřúhelníky - počty uzlů

Rozdělení uzlů do komunit podle grafu počtu uzlů na obrázku 13 vypadá téměř stejně jako v případě trojúhelníkl. Nyní pomocí grafu F-Score rozhodneme, nakolik je toto rozdělení podobné s referenčním rozdělením.



Obrázek 14: Radicchi, čtyřúhelníky - F-Score

Podle grafu na obrázku 14 se podobnost s referenčním rozdělení oproti verze algoritmu s trojúhelníky příliš neliší.
Vzhledem k podobným výsledným hodnotám s algoritmem vyhledávajícím trojúhelníky, zdá se, že u tohoto grafu nezáleží, zda hledáme trojúhelníky nebo čtyřúhelníky. Tato metoda se hodí do stejných situací jako metoda s trojúhelníky, akorát samozřejmě tuto metodu chceme použít na grafy obsahující velké množství čtyřúhelníků.

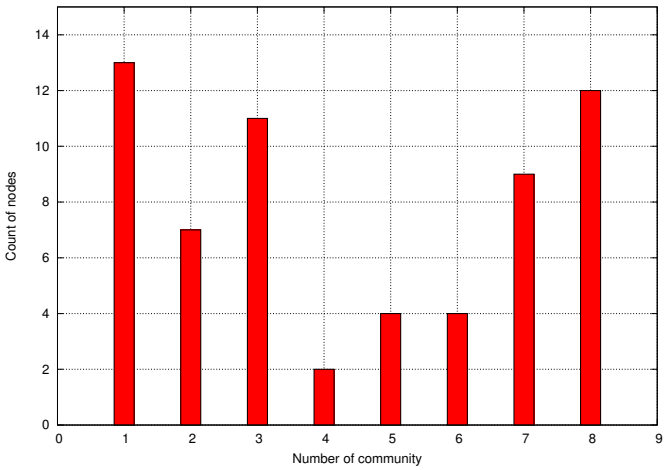
6.2.6 Dijkstra - jedna nejkratší cesta

Metoda, využívající k hledání nejkratších cest Dijkstrův algoritmus, slouží k vyhledávání komunit pro ohodnocené grafy. Aby jsem mohli tuto metodu porovnat, ke všem hranám testovaných dat byla přiřazena hodnota 1. Tentokrát testujeme metodu hledající pouze jednu nejkratší cestu. Takto můžeme simulovat výsledek pro neohodnocený graf. Hodnoty jsou v tabulce 9.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	13	7	11	2	4	4	9	12
Max. F-Score:	0.67	0.61	0.96	1	0.33	0.38	0.88	0.64
Odpovídající komunita:	1	2	3	4	5	6	7	8

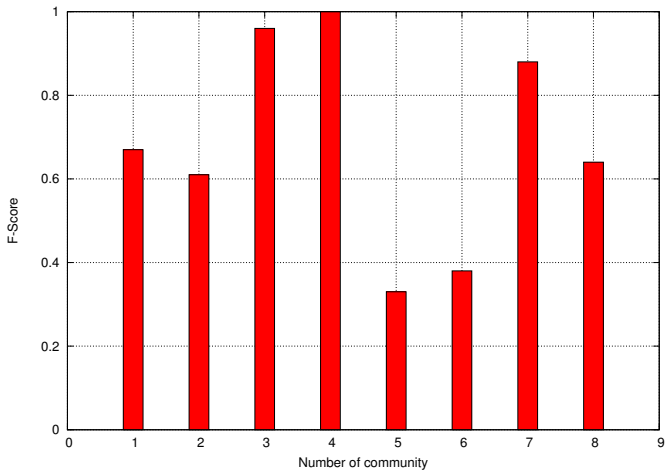
Tabulka 9: Dijkstra - jedna nejkratší cesta

Při pohledu na tabulku 9 se zdá být vše v pořádku, nenachází se zde žádná obrovská komunita. Nyní se podíváme na grafy.



Obrázek 15: Dijkstra, jedna cesta - počty uzlů

Rozdělení na obrázku 15 vypadá opět velice rovnoměrně, dokonce rovnoměrněji než rozdělení referenční. Nyní zjistíme, nakolik jsou si komunity podobné s komunitami referenčního rozdělení.



Obrázek 16: Dijkstra, jedna cesta - F-Score

Nyní k obrázku 16. Nalezené F-Score jednotlivých komunit nejsou nějak významně nízké, avšak také nejsou významně vysoké. Dá se říct, že výsledky jsou relativně dobré, šest z osmi komunit dosahují hodnoty F-Score vyšší než 0,6.

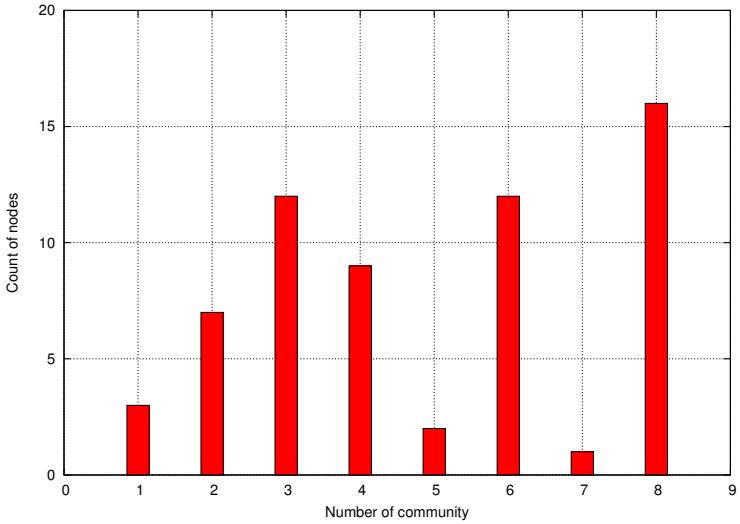
6.2.7 Dijkstra - více nejkratších cest

Tato verze Dijkstrovy metody vyhledává veškeré nejkratší cesty mezi dvojicemi uzlů, díky tomu je mírně časově náročnější. Nyní zkusíme otestovat, zda se tento nárůst času vyplácí. Vypočtená data jsou v tabulce 10.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	3	7	12	9	2	12	1	16
Max. F-Score:	1	0.6	1	0.88	1	0.86	0.11	0.97
Odpovídající komunita:	2	6	3	4	5	1	7	8

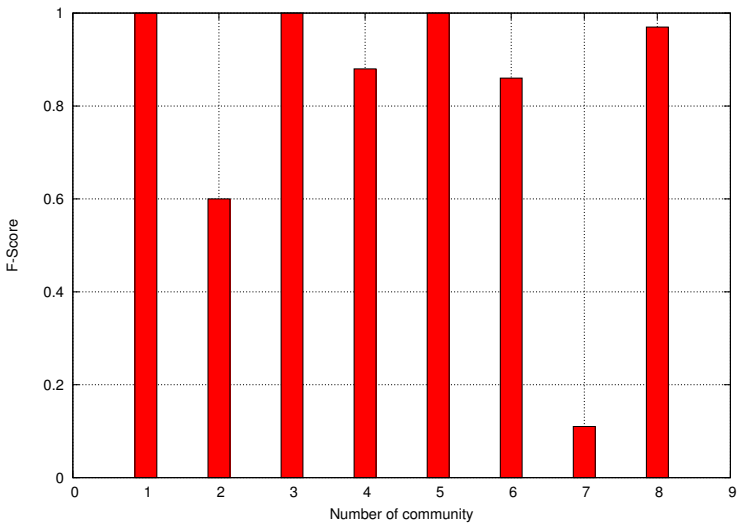
Tabulka 10: Dijsktra - více nejkratších cest

Počty uzlů v komunitách se zdají podobné jako u předešlé verze algoritmu, avšak komunita 7 má F-Score pouze 0,11. Zkusíme tedy z grafů vyčíst, zda je to problém.



Obrázek 17: Dijskra, více cest - počty uzlů

Tak podle grafu na obrázku 17 jde vidět, že naproti původním odhadům, tentokrát se nejedná o rovnoměrné rozdělení uzlů. Rozdělení daleko více odpovídá referenčnímu oddělení obsahující 4 velké a 4 malé komunity.



Obrázek 18: Dijskra, více cest - F-Score

Jak jsme odhadovali z grafu počtu uzlů, tak podle grafu na obrázku 18, je toto rozdělení skutečně velice podobné s rozdělením referenčním. Jediný problém vznikl u sedmé komunity, jejichž uzly byly přiřazeny mezi ostatní komunity a zbyl pouze jediný uzel. Bereme-li tedy referenční rozdělení jako ideální, pak je tato metoda daleko lepší, než Dijkstra využívající pouze jednu cestu. Dojde sice k nárůstu časové náročnosti, avšak výsledky se zdají být daleko lepší. To však neznamená, potřebujeme-li rychlejší algoritmus, že nemůžeme použít metodu jedné cesty, která také vrací obstojné výsledky.

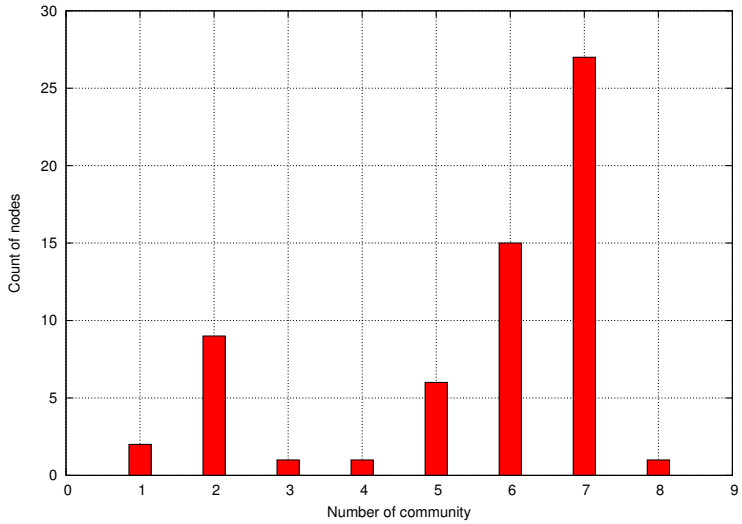
6.2.8 Path to zone

Tato metoda je ryze experimentální. Zatímco základy metody a myšlenky byly uvedeny v [9], avšak jak nakonec vypočíst hodnoty betweenness zde uvedeno nebylo. Pro otestování tedy byly vytvořeny dvě verze, kde druhá verze pouze doplňuje verzi první. Postup těchto algoritmů byl popsán v kapitolách betweenness hran a implementace. V tabulce 11 můžeme vidět výsledky první verze, a to verze path to zone.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	27	6	2	9	1	1	15	1
Max. F-Score:	0.64	0.55	0.21	0.88	0.15	0.5	0.65	0.11
Odpovídající komunita:	2	1	3	4	5	6	7	8

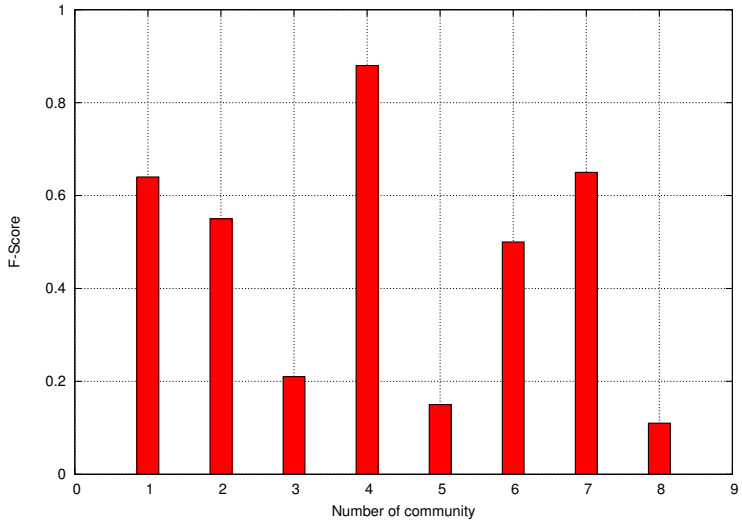
Tabulka 11: Path to zone

Při pohledu na tabulku vidíme, že vznikla jedna velká komunita s 27 uzly a to komunita 1. Z grafů zkusíme posoudit, jak se tato komunita projeví na zbytku komunit.



Obrázek 19: Path to zone - počty uzlů

Na obrázku 19 vidíme, že uzly se shromáždily do několika větších komunit, zatímco 4 menší komunity obsahují jeden či dva uzly. Toto rozdělení příliš neodpovídá referenčním komunitám.



Obrázek 20: Path to zone - F-Score

Podle grafu na obrázku 20 je vidět, že podobnost mezi tímto rozdělením a rozdělením referenčním, je velmi malá. Většinu větších F-Score hodnot dosahují pouze malé komunity. Jelikož se jednalo o experimentální metodu, nekvalitní výsledek příliš nepřekvapil. Vzhledem k časové náročnosti blížící se referenční metodě, skoro neušetříme ani dobu výpočtu. Z těchto důvodu se dá tato metoda označit jako selhání.

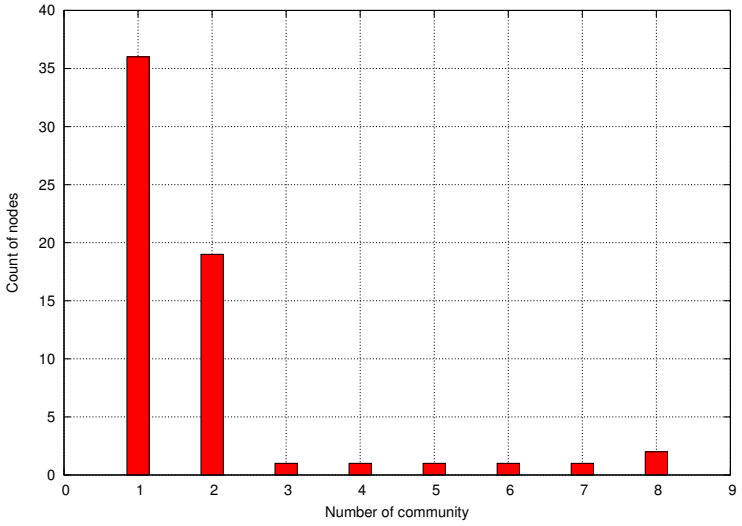
6.2.9 Path to zone - včetně vnitřku zón

Jelikož první verze této metody vrátila špatné výsledky, a tato meta je pomalejší než referenční, moc dobře to pro tuto metodu nevypadá. Tato verze metody navazuje na původní verzi, avšak prohledává graf více do hloubky. Naměřené hodnoty jsou v tabulce 12.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	36	19	1	1	1	1	1	2
Max. F-Score:	0.53	0.91	0.15	0.15	0.5	0.67	0.11	0.21
Odpovídající komunita:	2	6	3	4	5	1	7	8

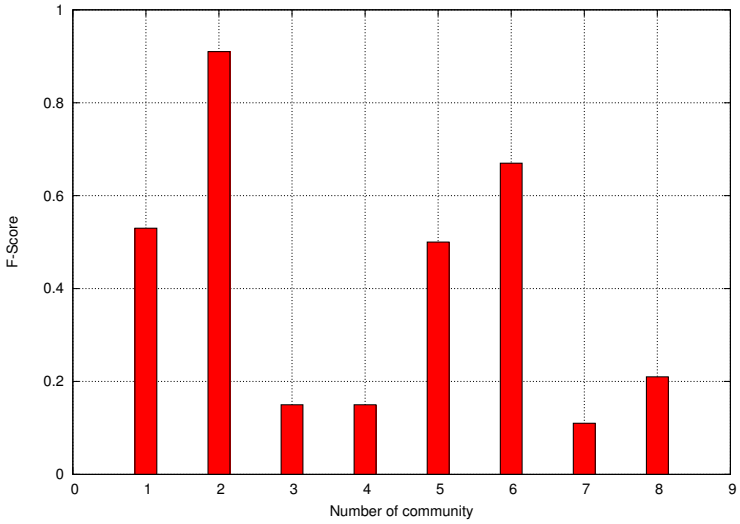
Tabulka 12: Path to zone - včetně vnitřku zón

Již při prvním pohledem na tabulku 12 jde vidět, že se jedná o selhání. V praxi byl graf rozdělen na dvě poloviny a poté díky podmínky dosažení osmi komunit, bylo z těchto komunit odloučeno několik uzlů.



Obrázek 21: Path to zone,včetně vnitřku zón - počty uzlů

Z grafu rozdělení uzlů na obrázku 21 jde vidět, že toto rozdělení absolutně neodpovídá referenčnímu rozdělení.



Obrázek 22: Path to zone,včetně vnitřku zón - F-Score

Hodnoty F-Score, jak můžeme vidět v grafu vypočtených hodnot na obrázku 22, jsou snad nejhorší, jaké byly doposud v této práci naměřeny. Vzhledem k rozdělení uzlů se tomuto výsledku ani nemůžeme divit.
Oproti první verzi této metody, tato verze má dokonce vyšší časovou náročnost, než referenční metoda, přitom výsledky jsou velice špatné. Tímto tedy obě verze experimentální metody path to zone se zdají být selhání. Jestli tomu skutečně je tak uvidíme při testování druhé datové kolekce.

6.3 Testování pomocí kolekce LesMiserables

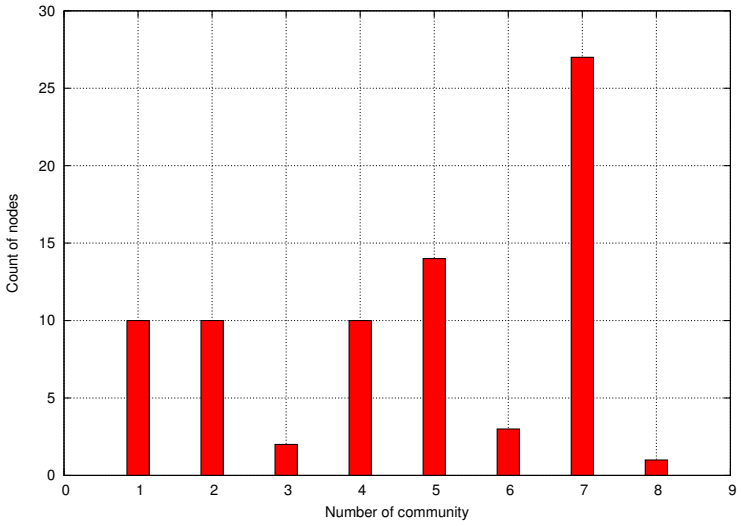
6.3.1 Referenční rozdělení

Datová kolekce LesMiserables je pomocí referenční metody přerozdělena do komunit uvedených v tabulce 13.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	10	2	10	14	3	27	1

Tabulka 13: Referenční rozdělení uzlů

Podle grafu rozdělení uzlů do komunit na obrázku 23 je vidět, že v počtu hran dominuje komunita 7. Dále byly vytvořeny 4 komunity o velikosti kolem desíti uzlů. Nakonec vznikly 3 komunity zanedbatelné velikosti. Když jsme se již seznámili s referenčním rozdělení uzlů, je čas podívat se, jaké rozdělení uzlů do komunit vznikly pomocí testovaných metod.



Obrázek 23: Referenční rozdělení uzlů

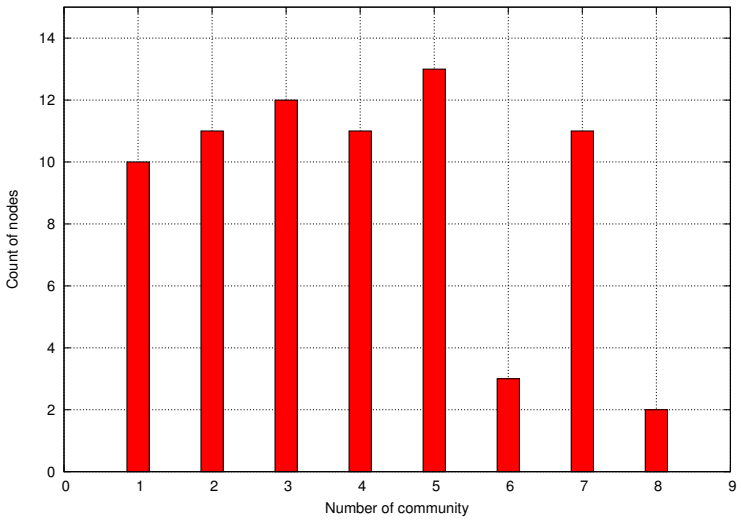
6.3.2 Nejkratší cesta - jedna nejkratší cesta

Opět začneme pouze úpravou referenční metody, tedy metodou vyhledávání pouze jedné nejrkatší cesty mezi uzly. Spočítaná data jsou v tabulce 14.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	11	12	11	13	3	11	2
Max. F-Score:	1	0.9	0.56	0.95	0.65	1	0.67	1
Odpovídající komunita:	1	4	2	3	5	6	7	8

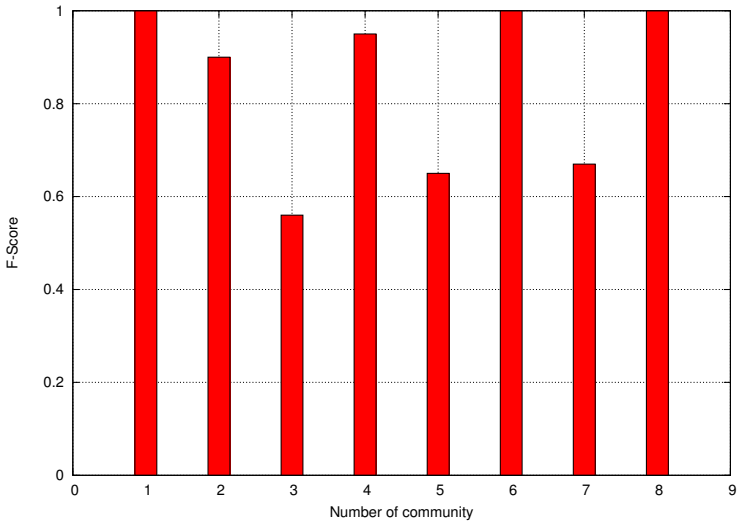
Tabulka 14: Nejkratší cesta - jedna cesta

Graf počtu uzlů na obrázku 24 vypovídá o více rovnoměrném rozdělení uzlů mezi komunitami. Část uzlů komunity 7 byla přerozdělena mezi ostatní komunity, což zajisté ovlivní výsledky F-Score.



Obrázek 24: Nejkratší cesta, jedna cesta - počty uzlů

F-Score dopadlo stejně jako u předešlé datové kolekce velice dobře. Dalo by se odhadnout, že nižší hodnoty některých komunit jsou způsoby již zmíněným přesunem uzlů z komunity 7 do komunit 3 a 5.



Obrázek 25: Nejkratší cesta, jedna cesta - F-Score

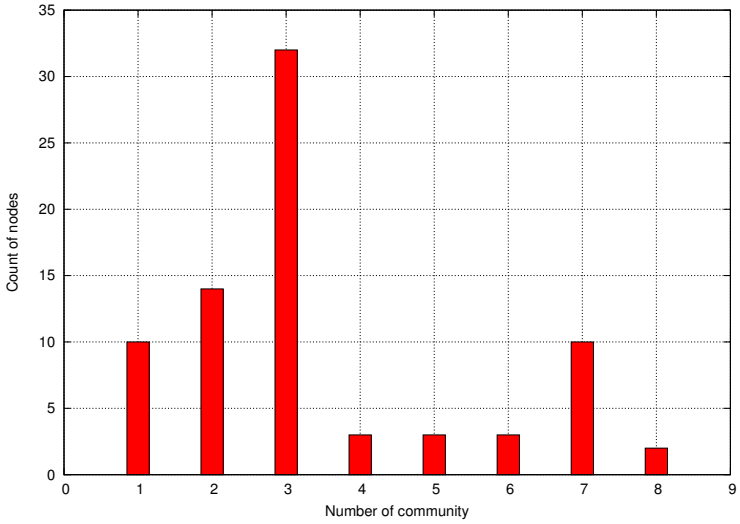
6.3.3 Náhodná procházka

Nyní opětovně otestujeme kvalitu rozdělení uzlů pomocí algoritmu náhodné procházky. Při testování pomocí kolekce Dolphins byly výsledky velmi dobré. Naměřené hodnoty jsou v tabulce 15.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	14	32	3	3	3	10	2
Max. F-Score:	1	1	0.92	0.46	0.46	1	1	1
Odpovídající komunita:	1	3	4	6	2	7	5	8

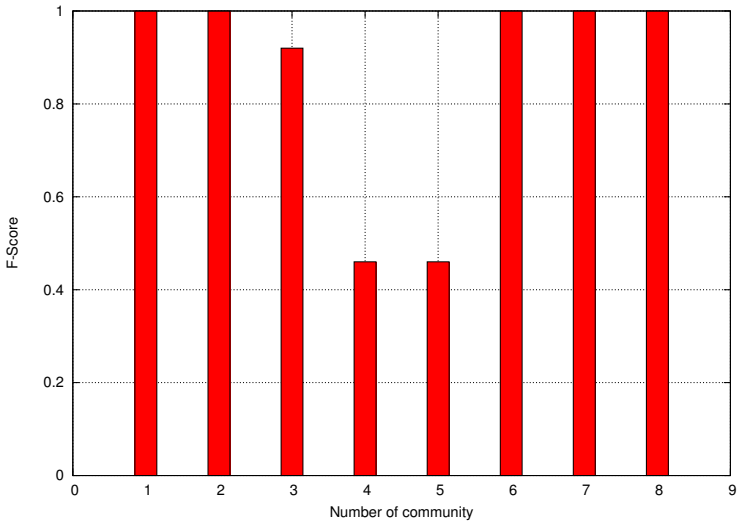
Tabulka 15: Náhodná procházka

Podle grafu rozdělení uzlů do komunit na obrázku 26 se zdá, že se rozdělení opět podobá referenčnímu rozdělení uzlů. Komunita s největším množstvím uzlů je však tentokrát ještě větší, nejspíše na úkor jedné ze 4 komunit, jenž dosahovaly velikosti kolem 10 uzlů.



Obrázek 26: Náhodná procházka - počty uzlů

Hodnoty F-Score na obrázku 27 jsou skutečně velice vysoké. U dvou komunit však došlo k značným rozdílům, lze tedy předpokládat, že tyto dvě komunity si navzájem promíchaly uzly. Rozdělení uzlů je i přes tyto dvě nízké hodnoty velice přesné.



Obrázek 27: Náhodná procházka - F.Score

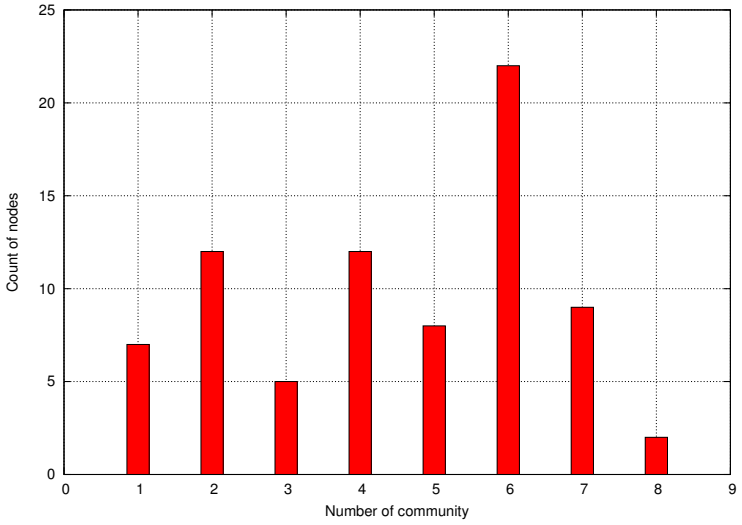
6.3.4 Radicchi - trojúhelníky

Tato metody nebyla během testování na datové kolekci Dolphins příliš přesná. Podíváme se, jak se ji bude dařit tentokrát. Vypočtené údaje se nacházejí v tabulce 16.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	8	7	5	12	12	22	9	2
Max. F-Score:	0.89	0.41	0.57	0.64	0.56	0.72	0.84	0.33
Odpovídající komunita:	2	1	4	3	6	7	5	8

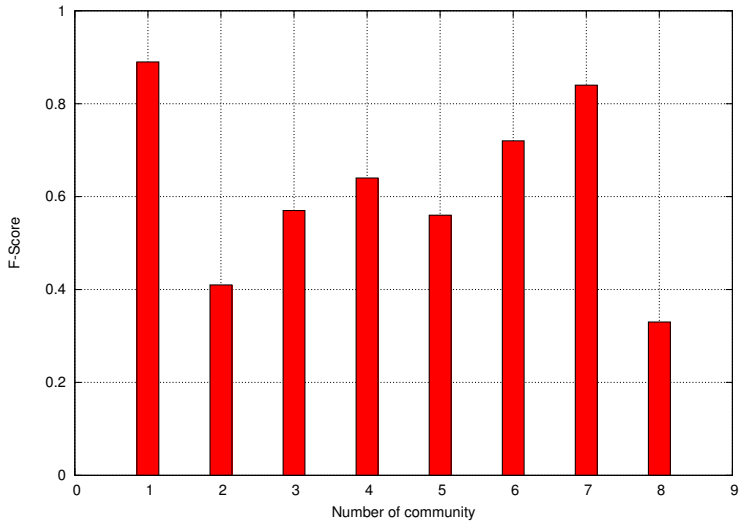
Tabulka 16: Radicchi - trojúhelníky

Podobně jako u referenčního rozdělení uzlů, i zde vznikla jedna velká komunita. Stejně tak vynikly čtyři komuniti velikosti 10 uzlů. Uvidíme, jak se toto odrazí na hodnotách F-Score.



Obrázek 28: Radicchi, trojúhelníky - počty uzlů

Podle grafu na obrázku 29 se zdá, že podobně jako při testování předešlé kolekce Dolphins, i tentokrát nebudou výsledky příliš přesné, avšak ani vyloženě špatné.



Obrázek 29: Radicchi, trojúhelníky - F.Score

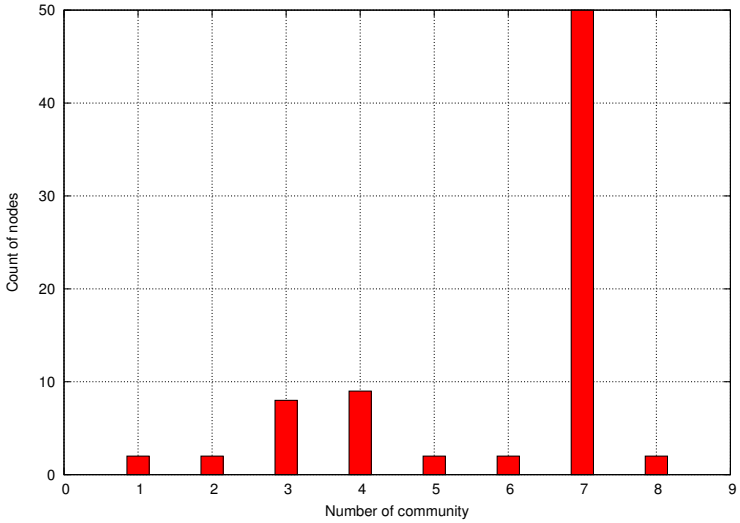
6.3.5 Radicchi - čtyřúhelníky

Nyní otestujeme druhou variantu Radicchi, která si u minulé kolekce vedla stejně jako varianta s trojúhelníky. Naměřené údaje v tabulce 17.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	8	2	2	50	9	2	2	2
Max. F-Score:	0.89	1	0.14	0.62	0.95	0.33	0.17	0.33
Odpovídající komunita:	4	1	3	5	6	2	7	8

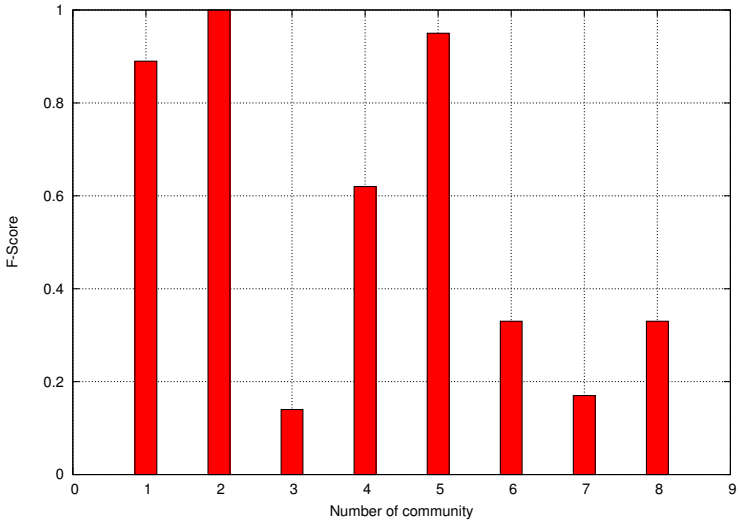
Tabulka 17: Radicchi - čtyřúhelníky

Podle obrázku 30 lze příliš odhadnout, že tentokrát dojde ke zhoršení přesnosti výsledků. Velikost největší komunity je 50 uzlů, což je více než polovina celkového počtu uzlů grafu. Z komunit velikosti 10 byly zachovány jen dvě.



Obrázek 30: Radicchi, čtyřúhelníky - počty uzlů

Hodnoty F-Score na obrázku 31 jsou jednoznačně horší, než výsledky varianty s trojúhelníky. Pouze miniaturní komunity maji dobré výsledky, zatímco komunita 7, obsahující většinu uzlů, má hodnotu F-Score pouze 0,17. Zdá se, že této datové kolekci metoda vyhledávající čtyřúhelníky nevyhovuje.



Obrázek 31: Radicchi, čtyřúhelníky - F.Score

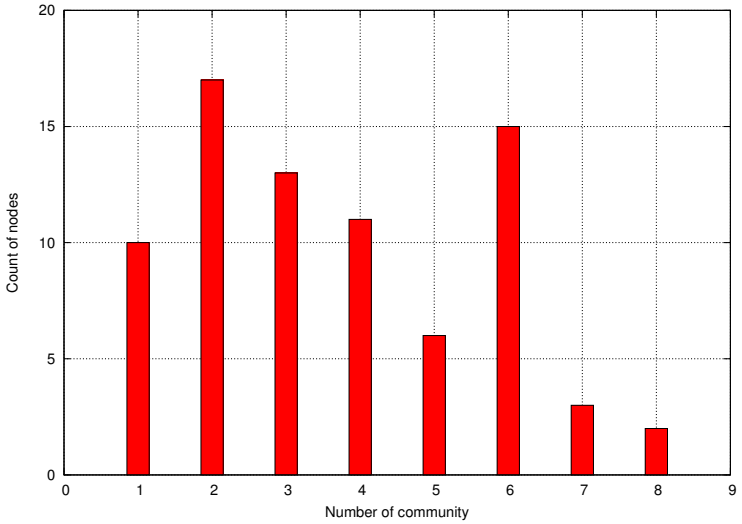
6.3.6 Dijkstra - jedna nejkratší cesta

Nyní bude otestována další varianta nejkratší cesty a to varianta pro grafy s ohodnocenými hranami. Data jsou v tabulce 18

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	17	13	11	6	15	3	2
Max. F-Score:	1	0.62	0.55	0.95	0.62	0.67	1	1
Odpovídající komunita:	1	2	4	3	6	5	7	8

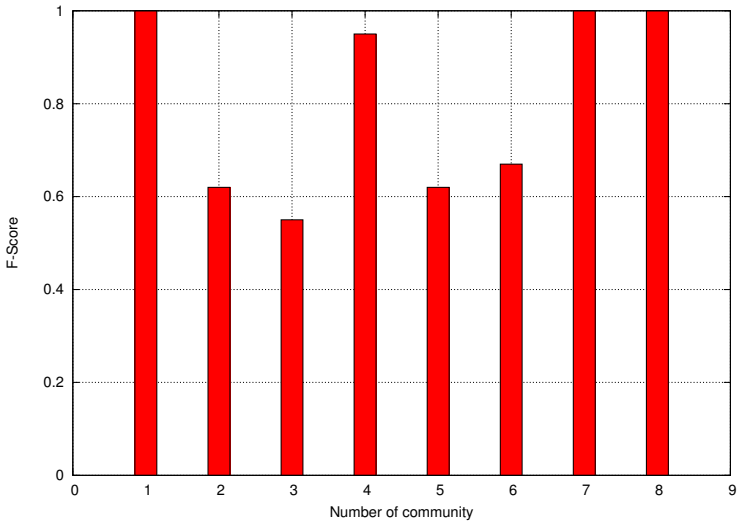
Tabulka 18: Dijkstra - jedna nejkratší cesta

Podle rozdělení uzlů na obrázku 32 se zdá být rozdělení oproti referenčnímu rozdělení značně odlišné. Nachází se spíše velký počet vyrovnaně velkých komunit.



Obrázek 32: Dijkstra, jedna cesta - počty uzlů

Oproti výsledkům grafu počtu uzlů, vypadají výsledky z graf F-Score na obrázku 33 docela dobře. Čtyři komunity mají mezi sebou promíchané uzly, jinak se výsledky shodují.



Obrázek 33: Dijkstra, jedna cesta - F.Score

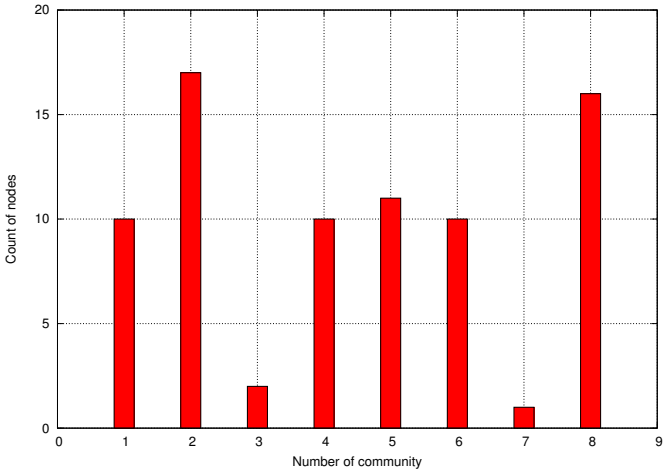
6.3.7 Dijkstra - Více nejkratších cest

Tato verze dijkstrova algoritmu měla u předešlé datové kolekce mírně lepší výsledky. Nyní otestujeme, jak to bude tentokrát. Data v tabulce 19.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	17	2	10	11	10	1	16
Max. F-Score:	1	0.9	1	1	0.58	1	0.07	0.8
Odpovídající komunita:	1	6	3	4	5	2	7	8

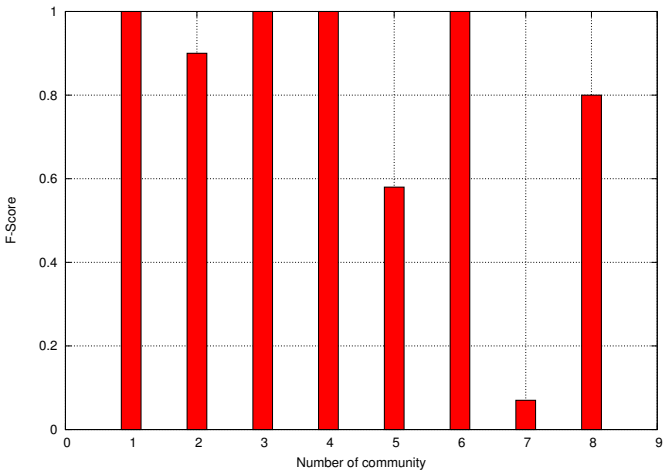
Tabulka 19: Dijkstra - více nejkratších cest

Podle obrázku 34 rozdělení uzlů mezi komunitami vypadá velice podobně jako u dřívější verze dijkstrova algoritmu. Opět se nedají odhadnout výsledky F-Score.



Obrázek 34: Dijkstra, více cest - počty uzlů

Podle obrázku 35 jsou výsledné hodnoty F-Score lepší, akorát komunita 7 má nízkou hodnotu. Tato komunita je ovšem tak malá, že nízká hodnota F-Score téměř nehraje roli.



Obrázek 35: Dijkstra, více cest - F-Score

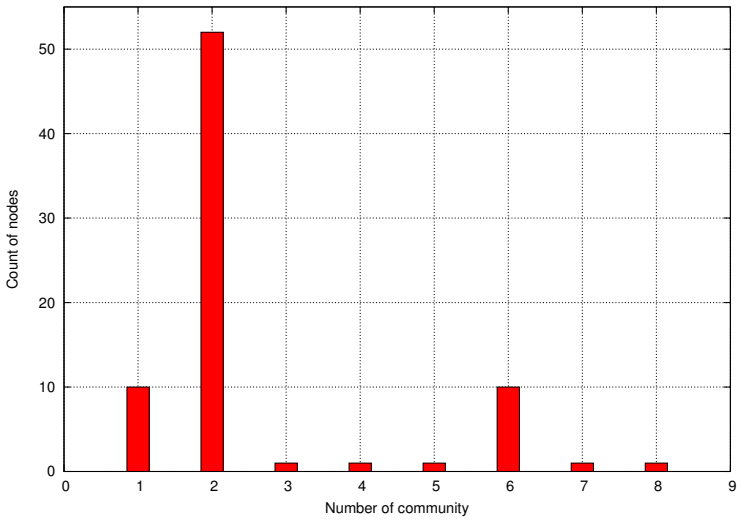
6.3.8 Path to zone

Při testování na datové kolekci Dolphins byly výsledky této metody velice špatné. Nyní se zkusíme ujistit, zda tato metoda skutečně vrací tak špatné výsledky. Vypočtená data v tabulce 20.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	52	1	1	1	10	1	1
Max. F-Score:	1	0.68	0.67	0.67	0.13	1	0.13	0.13
Odpovídající komunita:	1	5	3	4	6	2	7	8

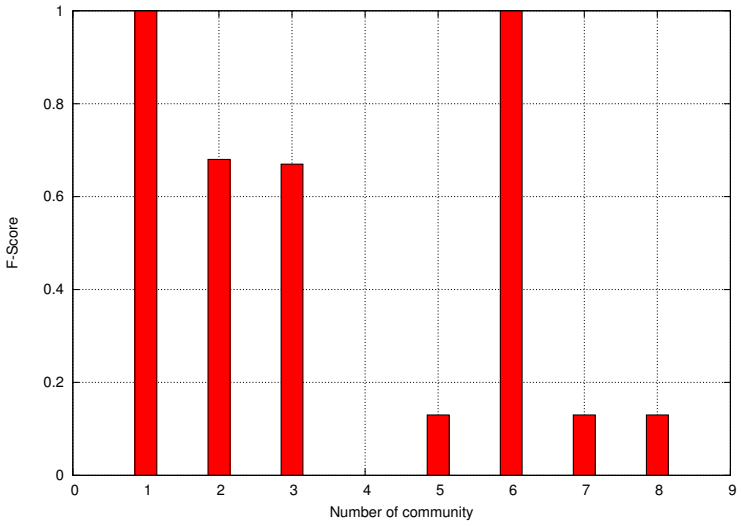
Tabulka 20: Path to Zone

Už při prvním pohledu na obrázek 36 můžeme vidět, že výsledky asi opět nebudou nejlepší. Nyní se tedy podíváme, jak jsou na tom hodnoty F-Score.



Obrázek 36: Path to zone - počty uzlů

Kupodivu dvě komunity mají s referenčními komunitami naprostou shodu. Zbývající hodnoty F-Score již tak dobré nejsou. Většina komunit ztratila některé uzly ve prospěch komunity 2. Potvrdilo se nám tedy, že tato metoda by v praxi neměla být používána.



Obrázek 37: Path to zone - F-Score

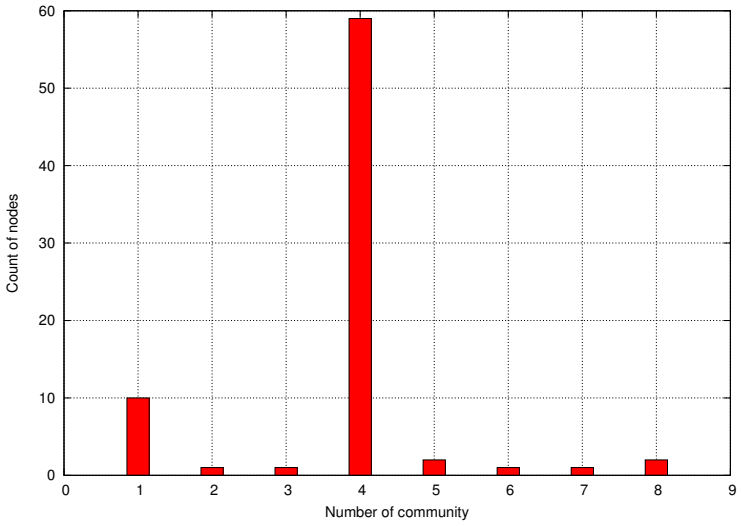
6.3.9 Path to zone - včetně vnitřku zón

Nakonec metoda, jenž u první datové kolekce dosáhla nejhorších výsledků, a to upravená metoda path to zone. Naměřené údaje v tabulce 21.

Id. komunity:	1	2	3	4	5	6	7	8
Počet uzlů:	10	1	1	59	2	1	1	2
Max. F-Score:	1	0.5	1	0.56	0.14	0.07	0.18	1
Odpovídající komunita:	4	1	3	5	6	2	7	8

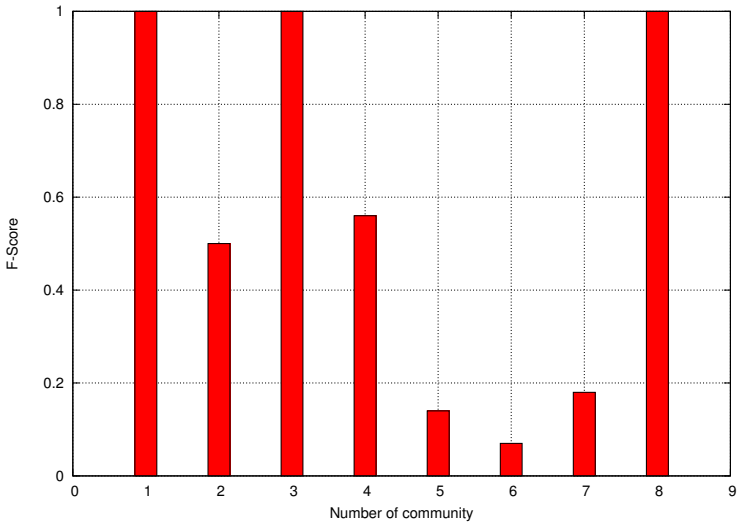
Tabulka 21: Path to Zone - včetně vnitřku zón

Graf na obrázku 38 vypadá velice podobně jako graf na obrázku 36, jedna komunita o velikosti 10 se však rozpadla. Můžeme tedy opět čekat ještě horších výsledků, než u předchozí metody.



Obrázek 38: Path to zone, včetně vnitřku zón - počty uzlů

Z obrázku 39 můžeme vyčíst, že komunita čtyři obsahuje množství uzlů ostatních komunit. Poněkud překvapivé je, že komunita 1 se opět naprosto shoduje s referenční komunitou. Opět jsme si potvrdili závěr výsledků testu s datovou kolekcí Dolphins, tedy že tato metoda by se v praxi neměla používat.



Obrázek 39: Path to zone, včetně vnitřku zón - F-Score

6.4 Vyhodnocení

Bylo provedeno 8 testů pro obě datové kolekce. Nyní tedy rekapitulujeme, co jsme se dozvěděli. Výsledky nejvíce podobné zvolenému referenčnímu rozdělení jsou výsledky metody nejkratší cesty a metody náhodné procházky. Náhodná procházka je však příliš pomalá, tak ji nelze doporučit k použití při řešení reálných problémů. Metoda využívající Dijkstrův algoritmus také vrátila dobré výsledky. Tato metoda je však vhodná pouze pro ohodnocené grafy. Špatně si taky nevedl Radicchi. Tato metoda je skutečně rychlá, avšak kvalita výsledků této metody příliš záleží na testovaném grafu a zvoleném podtypu metody. Nakonec byly otestovány experimentální metody path to zone. Obě varianty vrátily špatné výsledky, navíc se ani nepodařilo významně zkrátit dobu trvání algoritmu oproti metodě nejkratší cesty. Vzhledem k výsledkům testů, tyto metody nejsou vhodné pro reálné použití.

Jak tedy vyplývá z předešlého shrnutí, pro reálnou práci se z testovaných algoritmů nejvíce hodí algoritmus nejkratší cesty. V případě, že jsme si jistí, že graf splňuje požadavky algoritmu, vyplatí se použít algoritmus Radicchi. Pro ohodnocené grafy je nejvhodnější Dijsktrův algoritmus. Ostatní algoritmy nelze pro reálné použití doporučit.

7 Závěr

V rámci této práce byl vytvořen přehled, co to jsou grafy, k čemu v grafech slouží komunity a jak se tyto komunity dají nalézt. Dále bylo představeno několik metod k nalezení komunit, z nichž byl obzvlášť věnován prostor metodám vyhledávající hrany mezi komunitami.

Dále bylo naimplementováno několik vybraných metod, mezi nimi i metody experimentální. Díky četným článkům věnujícím se těmto metodám, během implementace téměř nedošlo k problémům. Také byly naimplementovány ukončující podmínky, které se svou podstatou dost lišily.

Nakonec byla implementována možnost provézt testy mezi referenčním výsledkem a výsledkem, jenž chceme otestovat. Tyto testy byly poté provedeny pro všechny naimplementované grafy a jejich výsledky byly sepsány v kapitole experimenty.

Jelikož se jednalo o práci přehledovou, snažící se pouze vytvořit přehled současných metod, může být tato práce považována za úspěch i přes špatné výsledky dvou experimentálních metod.

Program by mohl být rozšířen o další algoritmy pro vyhledávání komunit, nejen typu vyhledávající hrany mezi komunitami, nebo by mohla být implementována vizualizace grafu. Tyto prvky však již určitě existují v jiných, samostatných pracích.

Ondřej Svačina

8 Reference

[1] Markéta Brázdová, Využití některých metod teorie grafů při řešení dopravních problémů. 2007, [Cit. 29-03-2014]. Dostupné z: http://pernerscontacts.upce.cz/05 2007/Brazdova.pdf.

[2] FREEMAN, Linton C. The development of social network analysis: a study in the sociology of science. North Charleston, S.C.: BookSurge, 2004, xii, 205 p. ISBN 15-945-7714-5.

[3] PROULX, S, D PROMISLOW a P PHILLIPS. Network thinking in ecology and evolution. Trends in Ecology. 2005, vol. 20, issue 6, s. 345-353. DOI: 10.1016/j.tree.2005.04.004. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0169534705000881

[4] Ing. Vojtěch Hordějčuk, Teorie grafů. [Cit. 15-4-2014]. http://voho.cz/wiki/matematika/graf/.

[5] Rashid Bin Muhammad, Graph Theory. [Cit. 15- 4-2014]. http://www.personal.kent.edu/ rmuhamma/GraphTheory/MyGraphTheory-/defEx.htm.

[6] FORTUNATO, Santo. Community detection in graphs. Physics Reports. 2010, vol. 486, 3-5, s. 75-174. DOI: 10.1016/j.physrep.2009.11.002. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0370157309002841

[7] M. E. J. Newman, Scientific collaboration networks: II. Shortest paths, weighted networks, and centrality, Phys. Rev. E 64, 016132 2001.

[8] A note on two problems in connexion with graphs. Numerische Mathematik. 1959, vol. 1, issue 1, s. 269-271. DOI: 10.1007/BF01386390

[9] Rattigan, Matthew J.; Maier, Marc; and Jensen, David, "Using Structure Indices for Efficient Approximation of Network Properties"(2006). Computer Science Department Faculty Publication Series. Paper 166.

[10] M. E. J. Newman, and M. Girvan. Finding and evaluating community structure in networks Phys. Rev. E 69(2):026113 (February 2004)

[11] Xiang, Ju; Hu, Ke; Tang, Yi, A class of improved algorithms for detecting communities in complex networks. Physica A: Statistical Mechanics and its Applications, Volume 387, Issue 13, p. 3327-3334.

[12] Brandes, Ulrik (2008). "On variants of shortest-path betweenness centrality and their generic computation". Social Networks 30: 136–145. doi:10.1016/j.socnet.2007.11.001